cutadapt Documentation

Release 1.17

Marcel Martin

Aug 20, 2018

Contents

Cutadapt finds and removes adapter sequences, primers, poly-A tails and other types of unwanted sequence from your high-throughput sequencing reads.

Cleaning your data in this way is often required: Reads from small-RNA sequencing contain the 3' sequencing adapter because the read is longer than the molecule that is sequenced. Amplicon reads start with a primer sequence. Poly-A tails are useful for pulling out RNA from your sample, but often you don't want them to be in your reads.

Cutadapt helps with these trimming tasks by finding the adapter or primer sequences in an error-tolerant way. It can also modify and filter reads in various ways. Adapter sequences can contain IUPAC wildcard characters. Also, paired-end reads and even colorspace data is supported. If you want, you can also just demultiplex your input data, without removing adapter sequences at all.

Cutadapt comes with an extensive suite of automated tests and is available under the terms of the MIT license.

If you use cutadapt, please cite DOI:10.14806/ej.17.1.200 .

Links

- Documentation
- Source code
- Report an issue
- Project page on PyPI (Python package index)
- Follow @marcelm_ on Twitter
- Wrapper for the Galaxy platform

Table of contents

2.1 Installation

Cutadapt is being developed and tested under Linux. Users have run it successfully under macOS and Windows.

2.1.1 Quick installation

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

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

This will download the software from PyPI (the Python packaging index), and install the cutadapt binary into `$HOME/.local/bin`. If an old version of cutadapt exists on your system, the `--upgrade` parameter is required in order to install a newer version. 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.

2.1.2 Installation with conda

Alternatively, cutadapt is available as a conda package from the bioconda channel. If you do not have conda, install miniconda first. Then install cutadapt like this:

```
conda install -c bioconda cutadapt
```

If neither *pip* nor *conda* installation works, keep reading.

2.1.3 Dependencies

Cutadapt installation requires this software to be installed:

- Python 2.7 or at least Python 3.4

- Possibly a C compiler. For Linux, cutadapt packages are provided as so-called "wheels" (`.whl` files) which come pre-compiled.

Under Ubuntu, you may need to install the packages `build-essential` and `python-dev` (or `python3-dev`) to get a C compiler.

On Windows, you need Microsoft Visual C++ Compiler for Python 2.7.

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 the problem are missing Python development packages (`python-dev`/`python3-dev` in Ubuntu).

2.1.4 System-wide installation (root required)

If you have root access, then you can install cutadapt system-wide by running:

```
sudo pip install cutadapt
```

This installs cutadapt into */usr/local/bin*.

If you want to upgrade from an older version, use this command instead:

```
sudo pip install --upgrade cutadapt
```

2.1.5 Uninstalling

Type

```
pip uninstall cutadapt
```

and confirm with `y` to remove the package. Under some circumstances, multiple versions may be installed at the same time. Repeat the above command until you get an error message in order to make sure that all versions are removed.

2.1.6 Shared installation (on a cluster)

If you have a larger installation and want to provide cutadapt as a module that can be loaded and unloaded (with the Lmod system, for example), we recommend that you create a virtual environment and 'pip install' cutadapt into it. These instructions work on our SLURM cluster that uses the Lmod system (replace `1.9.1` with the actual version you want to use):

```
BASE=/software/cutadapt-1.9.1
virtualenv $BASE/venv
$BASE/venv/bin/pip install --install-option="--install-scripts=$BASE/bin"␣
→cutadapt==1.9.1
```

The `install-option` part is important. It ensures that a second, separate `bin/` directory is created (`/software/cutadapt-1.9.1/bin/`) that *only* contains the `cutadapt` script and nothing else. To make cutadapt available to the users, that directory (`$BASE/bin`) needs to be added to the `$PATH`.

Make sure you *do not* add the `bin/` directory within the `venv` directory to the `$PATH`! Otherwise, a user trying to run `python` who also has the cutadapt module loaded would get the python from the virtual environment, which leads to confusing error messages.

A simple module file for the Lmod system matching the above example could look like this:

```
conflict("cutadapt")
whatis("adapter trimming tool")
prepend_path("PATH", "/software/cutadapt-1.9.1/bin")
```

Please note that there is no need to "activate" the virtual environment: Activation merely adds the `bin/` directory to the `$PATH`, so the `prepend_path` directive is equivalent to activating the virtual environment.

2.1.7 Installing the development version

We recommend that you install cutadapt into a so-called virtual environment if you decide to use the development version. The virtual environment is a single directory that contains everything needed to run the software. Nothing else on your system is changed, so you can simply uninstall this particular version of cutadapt by removing the directory with the virtual environment.

The following instructions work on Linux using Python 3. Make sure you have installed the *dependencies* (`python3-dev` and `build-essential` on Ubuntu)!

First, choose where you want to place the directory with the virtual environment and what you want to call it. Let us assume you chose the path `~/cutadapt-venv`. Then use these commands for the installation:

```
python3 -m venv ~/cutadapt-venv
~/cutadapt-venv/bin/pip install Cython
~/cutadapt-venv/bin/pip install https://github.com/marcelm/cutadapt/archive/master.
↪zip
```

To run cutadapt and see the version number, type

```
~/cutadapt-venv/bin/cutadapt --version
```

The reported version number will be something like `1.14+65.g5610275`. This means that you are now running a cutadapt version that contains 65 additional changes (*commits*) since version 1.14.

2.2 User guide

2.2.1 Basic usage

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. You need to replace `AACCGGTT` with the correct adapter sequence. Reads are read from the input file `input.fastq` and are written to the output file `output.fastq`.

Compressed in- and output files are also supported:

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

Cutadapt searches for the adapter in all reads and removes it when it finds it. Unless you use a filtering option, 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.

Input and output file formats

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

- FASTA with extensions `.fasta`, `.fa` or `.fna`

- FASTQ with extensions `.fastq` or `.fq`

- Any of the above, but compressed as `.gz`, `.bz2` or `.xz`

Cutadapt's support for processing of colorspace data is described elsewhere.

Input and output file formats are recognized from the file name extension. You can override the input format with the `--format` option.

You can use the automatic format detection to convert from FASTQ to FASTA (without doing any adapter trimming):

```
cutadapt -o output.fasta.gz input.fastq.gz
```

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. This is why the example given above works:

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

All of cutadapt's options that expect a file name support this.

Files compressed with bzip2 (`.bz2`) or xz (`.xz`) are also supported, but only if the Python installation includes the proper modules. xz files require Python 3.3 or later.

Concatenated bz2 input files are *not supported* on Python versions before 3.3. These files are created by utilities such as `pbzip2` (parallel bzip2).

Concatenated gz input files *are* supported on all supported Python versions.

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
```

Wherever 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:

Chapter 2. Table of contents

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

Multi-core support

Cutadapt supports parallel processing, that is, it can use multiple CPU cores. Multi-core is currently not enabled by default. To enable it, use the option `-j N` (or the spelled-out version `--cores=N`), where `N` is the number of cores to use.

Make also sure that you have `pigz` (parallel gzip) installed if you use multiple cores and write to a `.gz` output file. Otherwise, compression of the output will be done in a single thread and therefore be the main bottleneck.

Note: In a future release, the plan is to make cutadapt automatically use as many CPU cores as are available, even when no `--cores` option was given. Please help to ensure that multi-core support is as stable as possible by reporting any problems you may find!

There are some limitations:

- Multi-core is *only* available when you run cutadapt with Python 3.3 or later.

- Multi-core cutadapt can only write to output files given by `-o` and `-p`. This implies that the following command-line arguments are not compatible with multi-core:

    - `--info-file`

    - `--rest-file`

    - `--wildcard-file`

    - `--untrimmed-output`, `--untrimmed-paired-output`

    - `--too-short-output`, `--too-short-paired-output`

    - `--too-long-output`, `--too-long-paired-output`

- Additionally, the following command-line arguments are not compatible with multi-core:

    - `--format`

    - `--colorspace`

- Multi-core is also not available when you use cutadapt for demultiplexing.

If you try to use multiple cores with an incompatible commandline option, you will get an error message.

Some of these limitations will be lifted in the future, as time allows.

New in version 1.15.

2.2.2 Read processing stages

Cutadapt can do a lot more in addition to removing adapters. There are various command-line options that make it possible to modify and filter reads and to redirect them to various output files. Each read is processed in the following order:

1. *Read modification options* are applied. This includes *adapter removal*, *quality trimming*, read name modifications etc. The order in which they are applied is the order in which they are listed in the help shown by `cutadapt --help` under the "Additional read modifications" heading. Adapter trimming itself does not appear in that list and is done after quality trimming and before length trimming (`--length`/`-l`).

2. *Filtering options* are applied, such as removal of too short or untrimmed reads. Some of the filters also allow to redirect a read to a separate output file. The filters are applied in the order in which they are listed in the help shown by `cutadapt --help` under the "Filtering of processed reads" heading.

3. If the read has passed all the filters, it is written to the output file.

2.2.3 Adapter types

Cutadapt can detect multiple adapter types. 5' adapters preceed the sequence of interest and 3' adapters follow it. Further distinctions are made according to where in the read the adapter sequence is allowed to occur to be found by the program.

Adapter type	Command-line option
Regular 3' adapter	`-a ADAPTER`
Regular 5' adapter	`-g ADAPTER`
Non-internal 3' adapter	`-a ADAPTERX`
Non-internal 5' adapter	`-g XADAPTER`
Anchored 3' adapter	`-a ADAPTER$`
Anchored 5' adapter	`-g ^ADAPTER`
5' or 3' (both possible)	`-b ADAPTER`
Linked adapter	`-a ADAPTER1...ADAPTER2`
Non-anchored linked adapter	`-g ADAPTER1...ADAPTER2`

By default, all adapters *are searched error-tolerantly*. Adapter sequences *may also contain any IUPAC wildcard character* (such as `N`).

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' and 5' ends.

Overview of the adapter types

3' adapter types

A 3' adapter is assumed to be ligated to the 3' end of your sequence of interest. When such an adapter is found, the adapter sequence itself and the sequence following it (if there is any) are trimmed. This table shows in which ways the different 3' adapter types are allowed to occur in a read in order to be recognized by the program.

Adapter location in read	Read layout	Found by regular 3' (`-a ADAPTER`)	Found by non-internal 3' (`-a ADAPTERX`)	Found by anchored 3' (`-a ADAPTER$`)
Full adapter sequence anywhere	acgtacg-tADAPTER-acgt	yes	no	no
Partial adapter sequence at 3' end	acgtacg-tacgtADAP	yes	yes	no
Full adapter sequence at 3' end	acgtacg-tacg-tADAPTER	yes	yes	yes

5' adapter types

A 5' adapter is assumed to be ligated to the 5' end of your sequence of interest. When such an adapter is found, the adapter sequence itself and the sequence preceding it (if there is any) are trimmed. This table shows in which ways the different 5' adapter types are allowed to occur in a read in order to be recognized by the program.

Adapter location in read	Read layout	Found by regular 5' (`-g ADAPTER`)	Found by non-internal 5' (`-g XADAPTER`)	Found by anchored 5' `-g ^ADAPTER`
Full adapter sequence anywhere	acg-tADAPTER-acgtacgt	yes	no	no
Partial adapter sequence at 5' end	PTERacg-tacgtacgt	yes	yes	no
Full adapter sequence at 5' end	ADAPTER-acgtacgtacgt	yes	yes	yes

Regular 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. By default, empty reads are kept and will appear in the output.

Regular 5' adapters

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

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.

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 beginning 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 merged, for example, overlapping paired-end reads, then it is 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 is 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.

Non-internal 5' and 3' adapters

The non-internal 5' and 3' adapter types disallow internal occurrences of the adapter sequence. This is like a less strict version of anchoring: The adapter must always be at one of the ends of the read, but - unlike anchored adapters - partial occurrences are also ok.

Use `-a ADAPTERX` (replace `ADAPTER` with your actual adapter sequence, but use a literal `X`) to disallow internal matches for a 3' adapter. Use `-g XADAPTER` to disallow them for a 5' adapter. Mnemonic: The `X` is not allowed to "shift into" the read.

Here are some examples for trimming reads with `-a ADAPTERX`:

Input read	Processed read
`mysequenceADAP`	`mysequence`
`mysequenceADAPTER`	`mysequence`
`mysequenceADAPTERsomethingelse`	`mysequenceADAPTERsomethingelse`

Here are some examples for trimming reads with `-g XADAPTER`:

Input read	Processed read
`APTERmysequence`	`mysequence`
`ADAPTERmysequence`	`mysequence`
`somethingelseADAPTERmysequence`	`somethingelseADAPTERmysequence`

New in version 1.17.

Linked adapters (combined 5' and 3' adapter)

If your sequence of interest ist "framed" by a 5' and a 3' adapter, and you want to remove both adapters, then you may want to use a *linked adapter*. A linked adapter combines an anchored 5' adapter and a 3' adapter. The 3' adapter can be regular or anchored. The idea is that a read is only trimmed if the anchored adapters occur. Thus, the 5' adapter is always required, and if the 3' adapter was specified as anchored, it also must exist for a successful match.

See the previous sections for what anchoring means.

Use `-a ADAPTER1...ADAPTER2` to search for a linked adapter. ADAPTER1 is always interpreted as an anchored 5' adapter. Here, ADAPTER2 is a regular 3' adapter. If you write `-a ADAPTER1...ADAPTER2$` instead, then the 3' adapter also becomes anchored, that is, for a read to be trimmed, both adapters must exist at the respective ends.

Note that the ADAPTER1 is always interpreted as an anchored 5' adapter even though there is no `^` character in the beginning.

In summary:

- `-a ADAPTER1...ADAPTER2`: The 5' adapter is removed if it occurs. If a 3' adapter occurs, it is removed only when also a 5' adapter is present.

- `-a ADAPTER1...ADAPTER2$`: The adapters are removed only if both occur.

As an example, assume the 5' adapter is *FIRST* and the 3' adapter is *SECOND* and you have these input reads:

```
FIRSTMYSEQUENCESECONDEXTRABASES
FIRSTMYSEQUENCESEC
FIRSTMYSEQUE
ANOTHERREADSECOND
```

Trimming with

```
cutadapt -a FIRST...SECOND -o output.fastq input.fastq
```

will result in

```
MYSEQUENCE
MYSEQUENCE
MYSEQUE
ANOTHERREADSECOND
```

The 3' adapter in the last read is not trimmed because the read does not contain the 5' adapter.

This feature does not work when used in combination with some other options, such as `--info-file`, `--mask-adapter`.

New in version 1.10.

New in version 1.13: Ability to anchor the 3' adapter.

Linked adapters without anchoring

This adapter type is especially suited for trimming CRISR screening reads.

Sometimes, the 5' adapter of a linked adapter pair should not be anchored. It is possible to specify linked adapters also with `-g ADAPTER1...ADAPTER2` (note that `-g` is used instead of `-a`). These work like the linked adapters described in the previous section, but with these two differences:

- The 5' adapter is not anchored by default. (So neither the 5' nor 3' adapter are anchored.)

- *Both* adapters are required. If one of them is not found, the read is not trimmed.

That is, when you use the *–discard-untrimmed'* option (or `--trimmed-only`) with a linked adapter specified with `-g`, then a read is considered to be trimmed if *both* adapter parts (5' and 3') are present in the read. This is different from linked adapters specified with `-a`, where a non-anchored 3' adapter is optional.

This feature has been added on a tentative basis. It may change in the next program version.

New in version 1.13.

Changed in version 1.15: Require both adapters for a read to be trimmed.

Linked adapter statistics

For linked adapters, the statistics report contains a line like this:

```
=== Adapter 1 ===

Sequence: AAAAAAAAA...TTTTTTTTTT; Type: linked; Length: 9+10; Trimmed: 3 times;
→Half matches: 2
```

The value for "Half matches" tells you how often only the 5'-side of the adapter was found, but not the 3'-side of it. This applies only to linked adapters with regular (non-anchored) 3' adapters.

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 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 cannot be used 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*.

Multiple adapter occurrences within a single read

If a single read contains multiple copies of the same adapter, the basic rule is that the leftmost match is used for both 5' and 3' adapters. For example, when searching for a 3' adapter in

```
cccccADAPTERgggggADAPTERttttt
```

the read will be trimmed to

```
ccccc
```

When the adapter is a 5' adapter instead, the read will be trimmed to

```
gggggADAPTERttttt
```

The above applies when both occurrences of the adapter are *exact* matches, and it also applies when both occurrences of the adapter are *inexact* matches (that is, it has at least one indel or mismatch). However, if one match is exact, but the other is inexact, then the exact match wins, even if it is not the leftmost one! The reason for this behavior is that cutadapt searches for exact matches first and, to improve performance, skips the error-tolerant matching step if an exact match was found.

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 parameter `--overlap` (or its short version `-O`). 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 zero, 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
```

Even the `X` wildcard that does not match any nucleotide is supported. If used as in `-a ADAPTERX` or `-g XADAPTER`, it acquires a special meaning for *and disallows internal adapter matches*.

Wildcard characters are by default only allowed in adapter sequences and are not recognized when they occur in a read. This is to avoid matches in reads that consist of many (often low-quality) `N` bases. Use `--match-read-wildcards` to enable wildcards also in reads.

Use the option `-N` to disable interpretation of wildcard characters even in the adapters. If wildcards are disabled entirely, that is, when you use `-N` and *do not* use `--match-read-wildcards`, then cutadapt compares characters by their 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.

Repeated bases in the adapter sequence

If you have many repeated bases in the adapter sequence, such as many `N` s or many `A` s, you do not have to spell them out. For example, instead of writing ten `A` in a row (`AAAAAAAAAA`), write `A{10}` instead. The number within the curly braces specifies how often the character that preceeds it will be repeated. This works also for IUPAC wildcard characters, as in `N{5}`.

It is recommended that you use quotation marks around your adapter sequence if you use this feature. For poly-A trimming, for example, you would write:

```
cutadapt -a "A{100}" -o output.fastq input.fastq
```

2.2.4 Modifying reads

This section describes in which ways reads can be modified other than adapter removal.

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 `--cut` is applied *before* any adapter trimming.

Quality trimming

The `-q` (or `--quality-cutoff`) 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.

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

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

By default, only the 3' end of each read is quality-trimmed. If you want to trim the 5' end as well, use the `-q` option with two comma-separated cutoffs:

```
cutadapt -q 15,10 -o output.fastq input.fastq
```

The 5' end will then be trimmed with a cutoff of 15, and the 3' end will be trimmed with a cutoff of 10. If you only want to trim the 5' end, then use a cutoff of 0 for the 3' end, as in `-q 10,0`.

A *description of the quality-trimming algorithm is also available*. The algorithm is the same as used by BWA.

Quality trimming of reads using two-color chemistry (NextSeq)

Some Illumina instruments use a two-color chemistry to encode the four bases. This includes the NextSeq and the (at the time of this writing) recently announced NovaSeq. In those instruments, a 'dark cycle' (with no detected color) encodes a `G`. However, dark cycles also occur when when sequencing "falls off" the end of the

fragment. The read then *contains a run of high-quality, but incorrect ''G' calls <*https://sequencing.qcfail.com/articles/illumina-2-colour-chemistry-can-overcall-high-confidence-g-bases/*>'_* at its 3' end.

Since the regular quality-trimming algorithm cannot deal with this situation, you need to use the `--nextseq-trim` option:

```
cutadapt --nextseq-trim=20 -o out.fastq input.fastq
```

This works like regular quality trimming (where one would use `-q 20` instead), except that the qualities of `G` bases are ignored.

New in version 1.10.

Shortening reads to a fixed length

To shorten each read down to a certain length, use the `--length` option or the short version `-l`:

```
cutadapt -l 10 -o output.fastq.gz input.fastq.gz
```

This shortens all reads from `input.fastq.gz` down to 10 bases. The removed bases are those on the 3' end.

If you want to remove a fixed number of bases from each read, use *the –cut option instead*.

Modifying read names

If you feel the need to modify the names of processed reads, some of the following options may be useful.

Use `-y` or `--suffix` to append a text to read names. The given string can contain the placeholder `{name}`, which will be replaced with the name of the adapter found in that read. For example, writing

```
cutadapt -a adapter1=ACGT -y ' we found {name}' input.fastq
```

changes a read named `read1` to `read1 we found adapter1` if the adapter `ACGT` was found. The options `-x`/`--prefix` work the same, but the text is added in front of the read name. For both options, spaces need to be specified explicitly, as in the above example. If no adapter was found in a read, the text `no_adapter` is inserted for `{name}`.

In order to remove a suffix of each read name, use `--strip-suffix`.

Some old 454 read files contain the length of the read in the name:

```
>read1 length=17
ACGTACGTACAAAAAAA
```

If you want to update this to the correct length after trimming, use the option `--length-tag`. In this example, this would be `--length-tag 'length='`. After trimming, the read would perhaps look like this:

```
>read1 length=10
ACGTACGTAC
```

Read modification order

The read modifications described above are applied in the following order to each read. Steps not requested on the command-line are skipped.

1. Unconditional base removal with `--cut`
2. Quality trimming (`-q`)
3. Adapter trimming (`-a`, `-b`, `-g` and uppercase versions)
4. Read shortening (`--length`)

5. N-end trimming (`--trim-n`)

6. Length tag modification (`--length-tag`)

7. Read name suffix removal (`--strip-suffix`)

8. Addition of prefix and suffix to read name (`-x/--prefix` and `-y/--suffix`)

9. Double-encode the sequence (only colorspace)

10. Replace negative quality values with zero (zero capping, only colorspace)

11. Trim primer base (only colorspace)

The last three steps are colorspace-specific.

2.2.5 Filtering reads

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 here make it possible to filter reads by either discarding them entirely or by redirecting them to other files. When redirecting reads, the basic rule 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 to which all modifications have been applied (adapter removal, quality trimming etc.). A processed read can be identical to the input read if no modifications were done.

`--minimum-length LENGTH or -m LENGTH` Discard processed reads that are shorter than LENGTH. Reads that are too short even before adapter removal are also discarded. Without this option, reads that have a length of zero (empty reads) are kept in the output.

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

`--maximum-length LENGTH or -M LENGTH` Discard processed reads that are longer than LENGTH. Reads that are too long even before adapter removal are also discarded.

`--too-long-output FILE` Instead of discarding 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` Discard reads in which an adapter was found.

`--discard-untrimmed` Discard reads 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 options `--untrimmed-output`, `--discard-trimmed` and `-discard-untrimmed` are mutually exclusive.

The following filtering options do not have a corresponding option for redirecting reads. They always discard reads for which the filtering criterion applies.

`--max-n COUNT_or_FRACTION` Discard reads with more than COUNT `N` bases. If COUNT_or_FRACTION is an number between 0 and 1, it is interpreted as a fraction of the read length

`--discard-casava` Discard reads that did not pass CASAVA filtering. Illumina's CASAVA pipeline in version 1.8 adds an *is_filtered* header field to each read. Specifying this option, the reads that did not pass filtering (these are the reads that have a `Y` for *is_filtered*) will be discarded. Reads for which the header cannot be recognized are kept.

2.2.6 Trimming paired-end reads

Cutadapt supports trimming of paired-end reads, trimming both reads in a pair at the same time.

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).

The basic command-line is:

```
cutadapt -a ADAPTER_FWD -A ADAPTER_REV -o out.1.fastq -p out.2.fastq reads.1.fastq
→reads.2.fastq
```

`-p` is the short form of `--paired-output`. The option `-A` is used here to specify an adapter sequence that cutadapt should remove from the second read in each pair. There are also the options `-G`, `-B`. All of them work just like their lowercase counterparts, except that the adapter is searched for in the second read in each paired-end read. There is also option `-U`, which you can use to remove a fixed number of bases from the second read in a pair.

While it is possible to run cutadapt on the two files separately, processing both files at the same time is highly recommended since the program can check for problems in your input files only when they are processed together.

When you use `-p`/`--paired-output`, cutadapt 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
```

This is an example for *improperly paired* read names:

```
@my_read/1;1
```

and:

```
@my_read/2;1
```

Since the `/1` and `/2` are ignored only if the occur at the end of the read name, and since the `;1` is considered to be part of the read name, these reads will not be considered to be propely paired.

As soon as you start to use one of the filtering options that discard reads, it is mandatory you process both files at the same time to make sure that the output files are kept synchronized: If a read is removed from one of the files, cutadapt will ensure it is also removed from the other file.

The following command-line options are applied to *both* reads:

- `-q` (along with `--quality-base`)
- `--times` applies to all the adapters given
- `--no-trim`
- `--trim-n`
- `--mask`
- `--length`
- `--length-tag`
- `--prefix`, `--suffix`
- `--strip-f3`
- `--colorspace`, `--bwa`, `-z`, `--no-zero-cap`, `--double-encode`, `--trim-primer`

The following limitations still exist:

- The `--info-file`, `--rest-file` and `--wildcard-file` options write out information only from the first read.

Filtering paired-end reads

The *filtering options listed above* can also be used when trimming paired-end data.

Importantly, cutadapt *always discards both reads of a pair* if it determines that the pair should be discarded. This ensures that the reads in the output files are in sync. (If you don't want or need this, you can run cutadapt separately on the R1 and R2 files.)

The same applies also to the options that redirect reads to other files if they fulfill a filtering criterion, such as `--too-short-output`/`--too-short-paired-output`. That is, the reads are always sent in pairs to these alternative output files.

By default, a read pair is discarded (or redirected) if one of the reads (R1 or R2) fulfills the filtering criterion. As an example, if option `--minimum-length=20` is used and paired-end data is processed, a read pair if discarded if one of the reads is shorter than 20 nt.

To require that filtering criteria must apply to *both* reads in order for a read pair to be discarded, use the option `--pair-filter=both`. The following table describes the effect for some filtering options.

Filtering option	With `--pair-filter=any`, the pair is discarded if ...	With `-pair-filter=both`, the pair is discarded if ...
`--minimum-length`	one of the reads is too short	both reads are too short
`--maximum-length`	one of the reads is too long	both reads are too long
`--discard-trimmed`	one of the reads contains an adapter	both reads contain an adapter
`--discard-untrimmed`	one of the reads does not contain an adapter	both reads do not contain an adapter
`--max-n`	one of the reads contains too many N bases	both reads contain too many N bases

To further complicate matters, cutadapt switches to a backwards compatibility mode ("legacy mode") when none of the uppercase modification options (`-A`/`-B`/`-G`/`-U`) are given. In that mode, filtering criteria are checked only for the *first* read. Cutadapt will also tell you at the top of the report whether legacy mode is active. Check that line if you get strange results!

These are the paired-end specific filtering and output options:

`--paired-output FILE` or `-p FILE` Write the second read of each processed pair to *FILE* (in FASTA/FASTQ format).

`--untrimmed-paired-output FILE` Used together with `--untrimmed-output`. The second read in a pair is written to this file when the processed pair was *not* trimmed.

`--too-short-paired-output FILE` Write the second read in a pair to this file if pair is too short. Use together with `--too-short-output`.

`--too-long-paired-output FILE` Write the second read in a pair to this file if pair is too long. Use together with `--too-long-output`.

`--pair-filter=(any|both)` Which of the reads in a paired-end read have to match the filtering criterion in order for it to be filtered.

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`.

Interleaved paired-end reads

Paired-end reads can be read from a single FASTQ file in which the entries for the first and second read from each pair alternate. The first read in each pair comes before the second. Enable this file format by adding the `--interleaved` option to the command-line. For example:

```
cutadapt --interleaved -q 20 -a ACGT -A TGCA -o trimmed.fastq reads.fastq
```

To read from an interleaved file, but write regular two-file output, provide the second output file as usual with the `-p` option:

```
cutadapt --interleaved -q 20 -a ACGT -A TGCA -o trimmed.1.fastq -p trimmed.2.fastq␣
↪reads.fastq
```

Reading two-file input and writing interleaved is also possible by providing a second input file:

```
cutadapt --interleaved -q 20 -a ACGT -A TGCA -o trimmed.1.fastq reads.1.fastq␣
↪reads.2.fastq
```

Cutadapt will detect if an input file is not properly interleaved by checking whether read names match and whether the file contains an even number of entries.

When `--interleaved` is used, legacy mode is disabled (that is, read-modification options such as `-q` always apply to both reads).

Legacy paired-end read trimming

Note: This section describes the way paired-end trimming was done in cutadapt before 1.8, where the `-A`, `-G`, `-B` options were not available. It is more complicated, but you can still use it.

If you do not use any of the filtering options that discard reads, such as `--discard`, `--minimum-length` or `--maximum-length`, you can 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
```

If you use any of the filtering options, you must use cutadapt in the following way (with the `-p` option) to make sure that read pairs remain sychronized.

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 reads.2.fastq
```

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.fastq tmp.1.fastq
```

Finally, remove the temporary files:

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

Please see the previous section for a much simpler way of trimming paired-end reads!

In legacy paired-end mode, the read-modifying options such as -q only apply to the first file in each call to cutadapt (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.

2.2.7 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 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. The file: syntax can be combined with the regular way of specifying an adapter. But no matter how you specify multiple adapter sequences, remember that 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*.

If you want to search for a combination of a 5' and a 3' adapter, you may want to provide them as a single so-called *"linked adapter"* instead.

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, which means that reads are 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}.fastq.gz input.fastq.gz
```

Here is a made-up example for the `barcodes.fasta` file:

```
>barcode01
TTAAGGCC
>barcode02
TAGCTAGC
>barcode03
ATGATGAT
```

Demultiplexing is also supported for paired-end data if you provide the `{name}` template in both output file names (`-o` and `-p`). Paired-end demultiplexing always uses the adapter matches of the *first* read to decide where a read should be written. If adapters to be found in read 2 are given (`-A`/`-G`), they are detected and removed as normal, but these matches do not influence where the read pair is written. This is to ensure that read 1 and read 2 are always synchronized. Example:

```
cutadapt -a first=AACCGG -a second=TTTTGG -A ACGTACGT -A TGCATGCA -o trimmed-{name}
→.1.fastq.gz -p trimmed-{name}.2.fastq.gz input.1.fastq.gz input.2.fastq.gz
```

This will create up to six output files named `trimmed-first.1.fastq.gz`, `trimmed-second.1.fastq.gz`, `trimmed-unknown.1.fastq.gz` and `trimmed-first.2.fastq.gz`, `trimmed-second.2.fastq.gz`, `trimmed-unknown.2.fastq.gz`.

You can use `--untrimmed-paired-output` to change the name for the output file that receives the untrimmed second reads.

New in version 1.15: Demultiplexing of paired-end data.

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 five copies of the adapter sequence. This command can be used to trim the reads correctly:

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

To search for a combination of a 5' and a 3' adapter, have a look at the *support for "linked adapters"* instead, which works better for that particular case because it is allows you to require that the 3' adapter is trimmed only when the 5' adapter also occurs, and it cannot happen that the same adapter is trimmed twice.

Before cutadapt supported linked adapters, the `--times` option was the recommended way to search for 5'/3' linked adapters. For completeness, we describe how it was done. 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 ...
```

2.2.8 Illumina TruSeq

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

Single-end reads as well as the first reads of paired-end data need to be trimmed 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.fastq.gz reads.fastq.gz
```

If you have paired-end data, trim also read 2 with the reverse complement of the "TruSeq Universal Adapter". The full command-line looks as follows:

```
cutadapt \
        -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC \
        -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT \
        -o trimmed.1.fastq.gz -p trimmed.2.fastq.gz \
        reads.1.fastq.gz reads.2.fastq.gz
```

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. However, you should be aware that this sequence occurs multiple times in the human genome and it could therefore skew your results very slightly at those loci

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

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

Under some circumstances you may want to consider not trimming adapters at all. If you have whole-exome or whole-genome reads, there will be very few reads with adapters anyway. And if you use BWA-MEM, the trailing (5') bases of a read that do not match the reference are soft-clipped, which covers those cases in which an adapter does occur.

Warning about incomplete adapter sequences

Sometimes cutadapt's report ends with these lines:

```
WARNING:
    One or more of your adapter sequences may be incomplete.
    Please see the detailed output above.
```

Further up, you'll see a message like this:

```
Bases preceding removed adapters:
  A: 95.5%
  C: 1.0%
  G: 1.6%
  T: 1.6%
  none/other: 0.3%
WARNING:
    The adapter is preceded by "A" extremely often.
    The provided adapter sequence may be incomplete.
    To fix the problem, add "A" to the beginning of the adapter sequence.
```

This means that in 95.5% of the cases in which an adapter was removed from a read, the base coming *before* that was an A. If your DNA fragments are not random, such as in amplicon sequencing, then this is to be expected and the warning can be ignored. If the DNA fragments are supposed to be random, then the message may be genuine: The adapter sequence may be incomplete and should include an additional A in the beginning.

This warning exists because some documents list the Illumina TruSeq adapters as starting with GATCGGA.... While that is technically correct, the library preparation actually results in an additional A before that sequence, which also needs to be removed. See the *previous section* for the correct sequence.

2.2.9 Dealing with N bases

Cutadapt supports the following options to deal with N bases in your reads:

--max-n COUNT Discard reads containing more than *COUNT* N bases. A fractional *COUNT* between 0 and 1 can also be given and will be treated as the proportion of maximally allowed N bases in the read.

--trim-n Remove flanking N bases from each read. That is, a read such as this:

```
NNACGTACGTNNNN
```

Is trimmed to just ACGTACGT. This option is applied *after* adapter trimming. If you want to get rid of N bases before adapter removal, use quality trimming: N bases typically also have a low quality value associated with them.

2.2.10 Bisulfite sequencing (RRBS)

When trimming reads that come from a library prepared with the RRBS (reduced representation bisulfite 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.

2.2.11 Cutadapt's output

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:
0-7 bp: 0; 8-15 bp: 1; 16-20 bp: 2
```

The adapter, as was shown above, has a length of 20 characters. We are using a custom error rate of 0.12. What this implies is shown above: Matches up to a length of 7 bp are allowed to have no errors. Matches of lengths 8-15 bp are allowd to have 1 error and matches of length 16 or more 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
7        13       0.3      0        1 12
8        31       0.1      1        0 31
...
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 0-7 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.

In the row for length 7 is an apparent anomaly, where the max.err column is 0 and yet we have 31 reads matching with 1 error. This is because the matches are actually contributed by alignments to the first 8 bases of the adapter with one deletion, so 7 bases are removed but the error cut-off applied is for length 8.

The "expect" column gives only a rough estimate of the number of sequences that is expected to match randomly, 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. For slightly more accurate estimates, you can provide the correct GC content (as a percentage) of your reads with the option `--gc-content`. The default is `--gc-content=50`.

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 (unless –*times* is used, see below). A row is written for *all* reads, even those that are discarded from the final output FASTA/FASTQ due to filtering options (such as `--minimum-length`).

The fields in each row 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.

9. Quality values corresponding to sequence left of the adapter match (can be empty)

10. Quality values corresponding to sequence matched to the adapter (can be empty)

11. Quality values corresponding to sequence to the right of the adapter match (can be empty)

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. Fields 9-11 are empty if quality values are not available. Concatenating them yields the full sequence of quality values.

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

1. Read name

2. The value -1

3. The read sequence

4. Quality values

When parsing the 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. Only for the first of those lines will the concatenation of columns 5-7 be identical to the original read sequence (and accordingly for columns 9-11). For subsequent lines, the shown sequence are the ones that were used in subsequent rounds of adapter trimming, that is, they get successively shorter.

New in version 1.9: Columns 9-11 were added.

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 (typically extensions `.csfasta` or `.csfa`)

- in a FASTQ file

- in a `.csfasta` and a `.qual` file (this is the native SOLiD format). That is, cutadapt expects *two* file names in this case.

In all cases, the colors must be represented by the characters 0, 1, 2, 3. Here is an example input file in `.fastq` format that is accepted:

```
@1_13_85_F3
T110020300.0113010210002110102330021
+
7&9<&77)& <7))%4'657-1+9;9,.<8);.;8
@1_13_573_F3
T312311200.30213011011113203302010003
+
6)3%)&&&& .1&(6:<'67..*,:75)'77&&&5
```

Further example input files can be found in the cutadapt distribution at `tests/data/solid.*`. The `.csfasta`/`.qual` file format is automatically assumed if two input files are given to cutadapt, and when no paired-end trimming options are used.

Cutadapt always converts input data given as a pair of FASTA/QUAL files to FASTQ.

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 `3302010303133112312` 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.fastq
```

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.

2.3.5 Sequence Read Archive

The Sequence Read Archive provides files in a special "SRA" file format. When the `fastq-dump` program from the sra-toolkit package is used to convert these `.sra` files to FASTQ format, colorspace reads will get an extra quality value in the beginning of each read. You may get an error like this:

```
cutadapt: error: In read named 'xyz': length of colorspace quality
sequence (36) and length of read (35) do not match (primer is: 'T')
```

To make cutadapt ignore the extra quality base, add `--format=sra-fastq` to your command-line, as in this example:

```
cutadapt -c --format=sra-fastq -a CGCCTTGGCCG sra.fastq > trimmed.fastq
```

When you use `--format=sra-fastq`, the spurious quality value will be removed from all reads in the file.

2.4 Algorithm details

2.4.1 Adapter 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. This means that mismatches, insertions and deletions are counted as one error, which is easier to understand and allows 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.

Quality trimming algorithm

The trimming algorithm implemented in cutadapt is the same as the one used by BWA, but applied to both ends of the read in turn (if requested). That is: Subtract the given cutoff from all qualities; compute partial sums from all indices to the end of the sequence; cut the sequence at the index at which the sum is minimal. If both ends are to be trimmed, repeat this for the other end.

The basic idea is to remove all bases starting from the end of the read whose quality is smaller than the given threshold. This is refined a bit by allowing some good-quality bases among the bad-quality ones. In the following example, we assume that the 3' end is to be quality-trimmed.

Assume you use a threshold of 10 and have these quality values:

42, 40, 26, 27, 8, 7, 11, 4, 2, 3

Subtracting the threshold gives:

32, 30, 16, 17, -2, -3, 1, -6, -8, -7

Then sum up the numbers, starting from the end (partial sums). Stop early if the sum is greater than zero:

(70), (38), 8, -8, -25, -23, -20, -21, -15, -7

The numbers in parentheses are not computed (because 8 is greater than zero), but shown here for completeness. The position of the minimum (-25) is used as the trimming position. Therefore, the read is trimmed to the first four bases, which have quality values 42, 40, 26, 27.

2.5 Recipes (FAQ)

This section gives answers to frequently asked questions. It shows you how to get cutadapt to do what you want it to do!

2.5.1 Remove more than one adapter

If you want to remove a 5' and 3' adapter at the same time, *use the support for linked adapters*.

If your situation is different, for example, when you have many 5' adapters but only one 3' adapter, then you have two options.

First, you can specify the adapters and also `--times=2` (or the short version `-n 2`). For example:

```
cutadapt -g ^TTAAGGCC -g ^AAGCTTA -a TACGGACT -n 2 -o output.fastq input.fastq
```

This instructs cutadapt to run two rounds of adapter finding and removal. That means that, after the first round and only when an adapter was actually found, another round is performed. In both rounds, all given adapters are searched and removed. The problem is that it could happen that one adapter is found twice (so the 3' adapter, for example, could be removed twice).

The second option is to not use the `-n` option, but to run cutadapt twice, first removing one adapter and then the other. It is easiest if you use a pipe as in this example:

```
cutadapt -g ^TTAAGGCC -g ^AAGCTTA input.fastq | cutadapt -a TACGGACT - > output.
↪fastq
```

2.5.2 Trim poly-A tails

If you want to trim a poly-A tail from the 3' end of your reads, use the 3' adapter type (-a) with an adapter sequence of many repeated A nucleotides. Starting with version 1.8 of cutadapt, you can use the following notation to specify a sequence that consists of 100 A:

```
cutadapt -a "A{100}" -o output.fastq input.fastq
```

This also works when there are sequencing errors in the poly-A tail. So this read

```
TACGTACGTACGTACGAAATAAAAAAAAAAA
```

will be trimmed to:

```
TACGTACGTACGTACG
```

If for some reason you would like to use a shorter sequence of A, you can do so: The matching algorithm always picks the leftmost match that it can find, so cutadapt will do the right thing even when the tail has more A than you used in the adapter sequence. However, sequencing errors may result in shorter matches than desired. For example, using -a "A{10}", the read above (where the AAAT is followed by eleven A) would be trimmed to:

```
TACGTACGTACGTACGAAAT
```

Depending on your application, perhaps a variant of -a A{10}N{90} is an alternative, forcing the match to be located as much to the left as possible, while still allowing for non-A bases towards the end of the read.

2.5.3 Trim a fixed number of bases after adapter trimming

If the adapters you want to remove are preceded by some unknown sequence (such as a random tag/molecular identifier), you can specify this as part of the adapter sequence in order to remove both in one go.

For example, assume you want to trim Illumina adapters preceded by 10 bases that you want to trim as well. Instead of this command:

```
cutadapt -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC ...
```

Use this command:

```
cutadapt -O 13 -a N{10}AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC ...
```

The -O 13 is the minimum overlap for an adapter match, where the 13 is computed as 3 plus 10 (where 3 is the default minimum overlap and 10 is the length of the unknown section). If you do not specify it, the adapter sequence would match the end of every read (because N matches anything), and ten bases would then be removed from every read.

2.5.4 Trimming (amplicon-) primers from both ends of paired-end reads

If you want to remove primer sequences that flank your sequence of interest, you should use a *"linked adapter"* to remove them. If you have paired-end data (with R1 and R2), you can correctly trim both R1 and R2 by using linked adapters for both R1 and R2. Here is how to do this.

The full DNA fragment that is put on the sequencer looks like this (looking only at the forward strand):

> 5' sequencing primer – forward primer – sequence of interest – reverse complement of reverse primer – reverse complement of 3' sequencing primer

Since sequencing of R1 starts after the 5' sequencing primer, R1 will start with the forward primer and then continue into the sequence of interest and into the two primers to the right of it, depending on the read length and how long the sequence of interest is. For R1, the linked adapter option that needs to be used is therefore

```
-a FWDPRIMER...RCREVPRIMER
```

where `FWDPRIMER` needs to be replaced with the sequence of your forward primer and `RCREVPRIMER` with the reverse complement of the reverse primer. The three dots `...` need to be entered as they are – they tell cutadapt that this is a linked adapter with a 5' and a 3' part.

Sequencing of R2 starts before the 3' sequencing primer and proceeds along the reverse-complementary strand. For the correct linked adapter, the sequences from above therefore need to be swapped and reverse-complemented:

```
-A REVPRIMER...RCFWDPRIMER
```

The uppercase `-A` specifies that this option is meant to work on R2. Similar to above, `REVPRIMER` is the sequence of the reverse primer and `RCFWDPRIMER` is the reverse-complement of the forward primer. Note that cutadapt does not reverse-complement any sequences of its own; you will have to do that yourself.

Finally, you may want to filter the trimmed read pairs. Use `--discard-untrimmed` to throw away all read pairs in which R1 doesn't start with `FWDPRIMER` or in which R2 does not start with `REVPRIMER`.

A note on how the filtering works: In linked adapters, by default the first part (before the `...`) is anchored. Anchored sequences *must* occur. If they don't, then the other sequence (after the `...`) is not even searched for and the entire read is internally marked as "untrimmed". This is done for both R1 and R2 and as soon as *any* of them is marked as "untrimmed", the entire pair is considered to be "untrimmed". If `--discard-untrimmed` is used, this means that the entire pair is discarded if R1 or R2 are untrimmed. (Option `--pair-filter=both` can be used to change this to require that *both* were marked as untrimmed.)

In summary, this is how to trim your data and discard all read pairs that do not contain the primer sequences that you know must be there:

```
cutadapt -a FWDPRIMER...RCREVPRIMER -A REVPRIMER...RCFWDPRIMER --discard-untrimmed
↪-o out.1.fastq.gz -p out.2.fastq.gz in.1.fastq.gz in.2.fastq.gz
```

2.5.5 Piping paired-end data

Sometimes it is necessary to run cutadapt twice on your data. For example, when you want to change the order in which read modification or filtering options are applied. To simplify this, you can use Unix pipes (`|`), but this is more difficult with paired-end data since then input and output consists of two files each.

The solution is to interleave the paired-end data, send it over the pipe and then de-interleave it in the other process. Here is how this looks in principle:

```
cutadapt [options] --interleaved in.1.fastq.gz in.2.fastq.gz | \
  cutadapt [options] --interleaved -o out.1.fastq.gz -p out.2.fastq.gz -
```

Note the `-` character in the second invocation to cutadapt.

2.5.6 Other things (unfinished)

- How to detect adapters
- Use cutadapt for quality-trimming only
- Use it for minimum/maximum length filtering
- Use it for conversion to FASTQ

2.6 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
- `--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
- instead of trimming, convert adapter to lowercase
- warn when given adapter sequence contains non-IUPAC characters
- try multithreading again, this time use os.pipe() or 0mq
- extensible file type detection
- the –times setting should be an attribute of Adapter

2.6.1 Backwards-incompatible changes

- Drop `--rest-file` support
- Possibly drop wildcard-file support, extend info-file instead
- Drop "legacy mode"
- For non-anchored 5' adapters, find rightmost match
- Move `scripts/cutadapt.py` to `__main__.py`

2.6.2 Specifying adapters

The idea is to deprecate the `-b`, `-g` and `-u` 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...` (or have anchoring by default?)
anywhere	`-b ADAPTER`	`-a ...ADAPTER... ???`
unconditional	`-u +10`	`-a 10...` (collides with colorspace)
unconditional	`-u -10`	`-a ...10$`
linked	`-a ADAPTER... ADAPTER`	`-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`.

The `...` would be equivalent to `N*` as in regular expressions.

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

Make it possible to specify that the rightmost or leftmost match should be picked. Default right now: Leftmost, even for -g adapters.

Allow `N{3,10}` as in regular expressions (for a variable-length sequence).

Use parentheses to specify the part of the sequence that should be kept:

- `-a (...)ADAPTER` (default)
- `-a (...ADAPTER)` (default)
- `-a ADAPTER(...)` (default)
- `-a (ADAPTER...)` (??)

Or, specify the part that should be removed:

> `-a ...(ADAPTER...) -a ...ADAPTER(...) -a (ADAPTER)...`

Model somehow all the flags that exist for semiglobal alignment. For start of the adapter:

- Start of adapter can be degraded or not
- Bases are allowed to be before adapter or not

Not degraded and no bases before allowed = anchored. Degraded and bases before allowed = regular 5'

By default, the 5' end should be anchored, the 3' end not.

- `-a ADAPTER...` → not degraded, no bases before allowed
- `-a N*ADAPTER...` → not degraded, bases before allowed
- `-a ADAPTER^...` → degraded, no bases before allowed
- `-a N*ADAPTER^...` → degraded, bases before allowed
- `-a ...ADAPTER` → degraded, bases after allowed
- `-a ...ADAPTER$` → not degraded, no bases after allowed

2.6.3 Paired-end trimming

- Could also use a paired-end read merger, then remove adapters with -a and -g

2.6.4 Available/used letters for command-line options

- Remaining characters: All uppercase letters except A, B, G, M, N, O, U
- Lowercase letters: i, j, k, s, w
- Planned/reserved: Q (paired-end quality trimming), j (multithreading)

2.7 Developing

The Cutadapt source code is on GitHub. Cutadapt is written in Python with some extension modules that are written in Cython. Cutadapt uses a single code base that is compatible with both Python 2 and 3. Python 2.7 is the minimum supported Python version.

2.7.1 Development installation

For development, make sure that you install Cython and tox. We also recommend using a virtualenv. This sequence of commands should work:

```
git clone https://github.com/marcelm/cutadapt.git  # or clone your own fork
cd cutadapt
virtualenv -p python3 venv  # or omit the "-p python3" for Python 2
venv/bin/pip3 install Cython pytest nose tox  # pip3 becomes just pip for Python 2
venv/bin/pip3 install -e .
```

Then you can run Cutadapt like this (or activate the virtualenv and omit the `venv/bin` part):

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

The tests can then be run like this:

```
venv/bin/pytest
```

Or with tox (but then you will need to have binaries for all tested Python versions installed):

```
venv/bin/tox
```

2.7.2 Development installation (without virtualenv)

Alternatively, if you do not want to use virtualenv, running the following may work from within the cloned repository:

```
python3 setup.py build_ext -i  # omit the "3" for Python 2
pytest
```

This requires Cython and pytest to be installed. Avoid this method and use a virtualenv instead if you can.

2.7.3 Code style

Cutadapt tries to follow PEP8, with some exceptions:

- Indentation is made with tabs, not with spaces
- The maximum line length for code 100 characters, not 80, but try to wrap comments at 80 characters for readability.

Yes, there are inconsistencies in the current code base since it's a few years old already.

2.7.4 Making a release

If this is the first time you attempt to upload a distribution to PyPI, create a configuration file named `.pypirc` in your home directory with the following contents:

```
[distutils]
index-servers =
    pypi

[pypi]
username=my-user-name
password=my-password
```

See also this blog post about getting started with PyPI. In particular, note that a `%` in your password needs to be doubled and that the password must *not* be put between quotation marks even if it contains spaces.

Cutadapt uses versioneer to automatically manage version numbers. This means that the version is not stored in the source code but derived from the most recent Git tag. The following procedure can be used to bump the version and make a new release.

1. Update `CHANGES.rst` (version number and list of changes)

2. Ensure you have no uncommitted changes in the working copy.

3. Run a `git pull`.

4. Run `tox`, ensuring all tests pass.

5. Tag the current commit with the version number (there must be a `v` prefix):

```
git tag v0.1
```

6. Create a distribution (`.tar.gz` file). Double-check that the auto-generated version number in the tarball is as you expect it by looking at the name of the generated file in `dist/`:

```
python3 setup.py sdist
```

7. If necessary, pip install `twine` and then upload the generated tar file to PyPI:

```
twine upload dist/cutadapt-0.1.tar.gz   # adjust version number
```

8. Push the tag:

```
git push --tags
```

9. Update the bioconda recipe. It is probly easiest to edit the recipe via the web interface and send in a pull request. Ensure that the list of dependencies (the `requirements:` section in the recipe) is in sync with the `setup.py` file.

   Since this is just a version bump, the pull request does not need a review by other bioconda developers. As soon as the tests pass and if you have the proper permissions, it can be merged directly.

If something went wrong *after* you uploaded a tarball, fix the problem and follow the above instructions again, but with an incremented revision in the version number. That is, go from version x.y to x.y.1. Do not change a version that has already been uploaded.

2.8 Changes

2.8.1 v1.17 (2018-08-20)

- Close #53: Implement adapters *that disallow internal matches*. This is a bit like anchoring, but less strict: The adapter sequence can appear at different lengths, but must always be at one of the ends. Use `-a ADAPTERX` (with a literal X) to disallow internal matches for a 3' adapter. Use `-g XADAPTER` to disallow for a 5' adapter.

- @klugem contributed PR #299: The `--length` option (and its alias `-l`) can now be used with negative lengths, which will remove bases from the beginning of the read instead of from the end.

- Close #107: Add a `--discard-casava` option to remove reads that did not pass CASAVA filtering (this is possibly relevant only for older datasets).

- Fix #318: Cutadapt should now be installable with Python 3.7.

- Running Cutadapt under Python 3.3 is no longer supported (Python 2.7 or 3.4+ are needed)

- Planned change: One of the next Cutadapt versions will drop support for Python 2 entirely, requiring Python 3.

2.8.2 v1.16 (2018-02-21)

- Fix #291: When processing paired-end reads with multiple cores, there could be errors about incomplete FASTQs although the files are intact.

- Fix #280: Quality trimming statistics incorrectly show the same values for R1 and R2.

2.8.3 v1.15 (2017-11-23)

- Cutadapt can now run on multiple CPU cores in parallel! To enable it, use the option `-j N` (or the long form `--cores=N`), where `N` is the number of cores to use. Multi-core support is only available on Python 3, and not yet with some command-line arguments. See *the new section about multi-core in the documentation* for details. When writing `.gz` files, make sure you have `pigz` installed to get the best speedup.

- The plan is to make multi-core the default (automatically using as many cores as are available) in future releases, so please test it and report an issue if you find problems!

- Issue #256: `--discard-untrimmed` did not have an effect on non-anchored linked adapters.

- Issue #118: Added support for demultiplexing of paired-end data.

2.8.4 v1.14 (2017-06-16)

- Fix: Statistics for 3' part of a linked adapter were reported incorrectly

- Fix issue #244: Quality trimming with `--nextseq-trim` would not apply to R2 when trimming paired-end reads.

- `--nextseq-trim` now disables legacy mode.

- Fix issue #246: installation failed on non-UTF8 locale

2.8.5 v1.13 (2017-03-16)

- The 3' adapter of linked adapters can now be anchored. Write `-a ADAPTER1...ADAPTER2$` to enable this. Note that the 5' adapter is always anchored in this notation.

- Issue #224: If you want the 5' part of a linked adapter *not* to be anchored, you can now write `-g ADAPTER...ADAPTER2` (note `-g` instead of `-a`). This feature is experimental and may change behavior in the next release.

- Issue #236: For more accurate statistics, it is now possible to specify the GC content of the input reads with `--gc-content`. This does not change trimming results, only the number in the "expect" column of the report. Since this is probably not needed by many people, the option is not listed when running `cutadapt --help`.

- Issue #235: Adapter sequences are now required to contain only valid IUPAC codes (lowercase is also allowed, `U` is an alias for `T`). This should help to catch hard-to-find bugs, especially in scripts. Use option `-N` to match characters literally (possibly useful for amino acid sequences).

- Documentation updates and some refactoring of the code

2.8.6 v1.12 (2016-11-28)

- Add read modification option `--length` (short: `-l`), which will shorten each read to the given length.

- Cutadapt will no longer complain that it has nothing to do when you do not give it any adapters. For example, you can use this to convert file formats: `cutadapt -o output.fasta input.fastq.gz` converts FASTQ to FASTA.

- The `xopen` module for opening compressed files was moved to a separate package on PyPI.

2.8.7 v1.11 (2016-08-16)

- The `--interleaved` option no longer requires that both input and output is interleaved. It is now possible to have two-file input and interleaved output, and to have interleaved input and two-file output.

- Fix issue #202: First and second FASTQ header could get out of sync when options modifying the read name were used.

2.8.8 v1.10 (2016-05-19)

- Added a new "linked adapter" type, which can be used to search for a 5' and a 3' adapter at the same time. Use `-a ADAPTER1...ADAPTER2` to search for a linked adapter. ADAPTER1 is interpreted as an anchored 5' adapter, which is searched for first. Only if ADAPTER1 is found will ADAPTER2 be searched for, which is a regular 3' adapter.

- Added experimental `--nextseq-trim` option for quality trimming of NextSeq data. This is necessary because that machine cannot distinguish between G and reaching the end of the fragment (it encodes G as 'black').

- Even when trimming FASTQ files, output can now be FASTA (quality values are simply dropped). Use the `-o`/`-p` options with a file name that ends in `.fasta` or `.fa` to enable this.

- Cutadapt does not bundle pre-compiled C extension modules (`.so` files) anymore. This affects only users that run cutadapt directly from an unpacked tarball. Install through `pip` or `conda` instead.

- Fix issue #167: Option `--quiet` was not entirely quiet.

- Fix issue #199: Be less strict when checking for properly-paired reads.

- This is the last version of cutadapt to support Python 2.6. Future versions will require at least Python 2.7.

2.8.9 v1.9.1 (2015-12-02)

- Added `--pair-filter` option, which *modifies how filtering criteria apply to paired-end reads*

- Add `--too-short-paired-output` and `--too-long-paired-output` options.

- Fix incorrect number of trimmed bases reported if `--times` option was used.

2.8.10 v1.9 (2015-10-29)

- Indels in the alignment can now be disabled for all adapter types (use `--no-indels`).

- Quality values are now printed in the info file (`--info-file`) when trimming FASTQ files. Fixes issue #144.

- Options `--prefix` and `--suffix`, which modify read names, now accept the placeholder `{name}` and will replace it with the name of the found adapter. Fixes issue #104.

- Interleaved FASTQ files: With the `--interleaved` switch, paired-end reads will be read from and written to interleaved FASTQ files. Fixes issue #113.

- Anchored 5' adapters can now be specified by writing `-a SEQUENCE...` (note the three dots).

- Fix `--discard-untrimmed` and `--discard-trimmed` not working as expected in paired-end mode (issue #146).

- The minimum overlap is now automatically reduced to the adapter length if it is too large. Fixes part of issue #153.

- Thanks to Wolfgang Gerlach, there is now a Dockerfile.

- The new `--debug` switch makes cutadapt print out the alignment matrix.

2.8.11 v1.8.3 (2015-07-29)

- Fix issue #95: Untrimmed reads were not listed in the info file.

- Fix issue #138: pip install cutadapt did not work with new setuptools versions.

- Fix issue #137: Avoid a hang when writing to two or more gzip-compressed output files in Python 2.6.

2.8.12 v1.8.2 (2015-07-24)

2.8.13 v1.8.1 (2015-04-09)

- Fix #110: Counts for 'too short' and 'too long' reads were swapped in statistics.

- Fix #115: Make `--trim-n` work also on second read for paired-end data.

2.8.14 v1.8 (2015-03-14)

- Support single-pass paired-end trimming with the new `-A/-G/-B/-U` parameters. These work just like their -a/-g/-b/-u counterparts, but they specify sequences that are removed from the *second read* in a pair.

  Also, if you start using one of those options, the read modification options such as `-q` (quality trimming) are applied to *both* reads. For backwards compatibility, read modifications are applied to the first read only if neither of `-A/-G/-B/-U` is used. See the documentation for details.

  This feature has not been extensively tested, so please give feedback if something does not work.

- The report output has been re-worked in order to accomodate the new paired-end trimming mode. This also changes the way the report looks like in single-end mode. It is hopefully now more accessible.

- Chris Mitchell contributed a patch adding two new options: `--trim-n` removes any N bases from the read ends, and the `--max-n` option can be used to filter out reads with too many N.

- Support notation for repeated bases in the adapter sequence: Write `A{10}` instead of `AAAAAAAAAA`. Useful for poly-A trimming: Use `-a A{100}` to get the longest possible tail.

- Quality trimming at the 5' end of reads is now supported. Use `-q 15,10` to trim the 5' end with a cutoff of 15 and the 3' end with a cutoff of 10.

- Fix incorrectly reported statistics (> 100% trimmed bases) when `--times` set to a value greater than one.

- Support .xz-compressed files (if running in Python 3.3 or later).

- Started to use the GitHub issue tracker instead of Google Code. All old issues have been moved.

2.8.15 v1.7 (2014-11-25)

- 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.

2.8.16 v1.6 (2014-10-07)

- 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.

- Due 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.

2.8.17 v1.5 (2014-08-05)

- 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.

2.8.18 v1.4 (2014-03-13)

- 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

2.8.19 v1.3 (2013-11-08)

- 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.

2.8.20 v1.2 (2012-11-30)

- 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.

2.8.21 v1.1 (2012-06-18)

- 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`)

2.8.22 v1.0 (2011-11-04)

- 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`.

2.8.23 v0.9.5 (2011-07-20)

- 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.

2.8.24 v0.9.4 (2011-05-20)

- 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.

2.8.25 v0.9.2 (2011-03-16)

- 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)

2.8.26 v0.9 (2011-01-10)

- 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

2.8.27 v0.8 (2010-12-08)

- 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).

2.8.28 v0.7 (2010-12-03)

- 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.

2.8.29 v0.6 (2010-11-18)

- Support gzipped input and output.

- Print timing information in statistics.

2.8.30 v0.5 (2010-11-17)

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

2.8.31 v0.4 (2010-11-17)

- (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.

2.8.32 v0.3 (2010-09-27)

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

2.8.33 v0.2 (2010-09-14)

- allow FASTQ input

2.8.34 v0.1 (2010-09-14)

- initial release