The GO *OBITools*

SEric Coissac

2022-08-25

Contents

Chapter 1

The OBITools

1.1 Aims of *OBITools*

1.2 File formats usable with *OBITools*

1.2.1 The sequence files

Sequences can be stored following various format. OBITools knows some of them. The central formats for sequence files manipulated by OBITools scripts are the `fasta` and fastq format. OBITools extends the both these formats by specifying a syntax to include in the definition line data qualifying the sequence. All file formats use the `IUPAC` code for encoding nucleotides.

1.2.2 The IUPAC Code

The International Union of Pure and Applied Chemistry (IUPAC_) defined the standard code for representing protein or DNA sequences.

1.2.2.1 Nucleic IUPAC Code

Code	**Nucleotide**
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
U	Uracil
R	Purine (A or G)
Y	Pyrimidine (C, T, or U)
M	C or A

Code	Nucleotide
K	T, U, or G
W	T, U, or A
S	C or G
B	C, T, U, or G (not A)
D	A, T, U, or G (not C)
H	A, T, U, or C (not G)
V	A, C, or G (not T, not U)
N	Any base (A, C, G, T, or U)

1.2.3 The *fasta* format

The **fasta format** is certainly the most widely used sequence file format. This is certainly due to its great simplicity. It was originally created for the Lipman and Pearson FASTA program. OBITools use in more of the classical :ref:`fasta` format an :ref:`extended version` of this format where structured data are included in the title line.

In *fasta* format a sequence is represented by a title line beginning with a `>` character and the sequences by itself following the :doc:`iupac` code. The sequence is usually split other severals lines of the same length (expect for the last one)

```
>my_sequence this is my pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
```

This is no special format for the title line excepting that this line should be unique. Usually the first word following the > character is considered as the sequence identifier. The end of the title line corresponding to a description of the sequence. Several sequences can be concatenated in a same file. The description of the next sequence is just pasted at the end of the record of the previous one

```
>sequence_A this is my first pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
>sequence_B this is my second pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
>sequence_C this is my third pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
```

1.2.4 The *fastq* sequence format

.. note::

```
This article uses material from the Wikipedia article
`FASTQ format `
which is released under the
`Creative Commons Attribution-Share-Alike License 3.0 `
```

fastq format is a text-based format for storing both a biological sequence (usually nucleotide sequence) and its corresponding quality scores. Both the sequence letter and quality score are encoded with a single ASCII character for brevity. It was originally developed at the `Wellcome Trust Sanger Institute` to bundle a fasta sequence and its quality data, but has recently become the *de facto* standard for storing the output of high throughput sequencing instruments such as the Illumina Genome Analyzer Illumina. [1]_

1.2.4.1 Format

A fastq file normally uses four lines per sequence.

- Line 1 begins with a '@' character and is followed by a sequence identifier and an *optional* description (like a :ref:`fasta` title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a '+' character and is *optionally* followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

A fastq file containing a single sequence might look like this:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

The character '!' represents the lowest quality while '~' is the highest. Here are the quality value characters in left-to-right increasing order of quality (`ASCII`):

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

The original Sanger FASTQ files also allowed the sequence and quality strings to be wrapped (split over multiple lines), but this is generally discouraged as it can make parsing complicated due to the unfortunate choice of "@" and "+" as markers (these characters can also occur in the quality string).

1.2.4.2 Variations

1.2.4.2.1 Quality A quality value Q is an integer mapping of p (i.e., the probability that the corresponding base call is incorrect). Two different equations have been in use. The first is the standard Sanger variant to assess reliability of a base call, otherwise known as Phred quality score:

$$Q_{\text{sanger}} = -10 \log_{10} p$$

The Solexa pipeline (i.e., the software delivered with the Illumina Genome Analyzer) earlier used a different mapping, encoding the odds $\mathbf{p}/(1-\mathbf{p})$ instead of the probability $\mathbf{p}$:

$$Q_{\text{solexa-prior to v.1.3}} = -10 \log_{10} \frac{p}{1-p}$$

Although both mappings are asymptotically identical at higher quality values, they differ at lower quality levels (i.e., approximately $\mathbf{p} > 0.05$, or equivalently, $\mathbf{Q} < 13$).

|Relationship between Q and p using the Sanger (red) and Solexa (black) equations (described above). The vertical dotted line indicates $\mathbf{p} = 0.05$, or equivalently, $Q = 13$.|

1.2.4.3 Encoding

- Sanger format can encode a Phred quality score from 0 to 93 using ASCII 33 to 126 (although in raw read data the Phred quality score rarely exceeds 60, higher scores are possible in assemblies or read maps).
- Solexa/Illumina 1.0 format can encode a Solexa/Illumina quality score from -5 to 62 using ASCII 59 to 126 (although in raw read data Solexa scores from -5 to 40 only are expected)
- Starting with Illumina 1.3 and before Illumina 1.8, the format encoded a Phred quality score from 0 to 62 using ASCII 64 to 126 (although in raw read data Phred scores from 0 to 40 only are expected).
- Starting in Illumina 1.5 and before Illumina 1.8, the Phred scores 0 to 2 have a slightly different meaning. The values 0 and 1 are no longer used and the value 2, encoded by ASCII 66 “B”.

Sequencing Control Software, Version 2.6, Catalog # SY-960-2601, Part # 15009921 Rev. A, November 2009] [http://watson.nci.nih.gov/solexa/Using_SCSv2.6_15009921_A.pdf\\]((page 30) states the following: *If a read ends with a segment of mostly low quality (Q15 or below), then all of the quality values in the segment are replaced with a value of 2 (encoded as the letter B in Illumina’s text-based encoding of quality scores)... This Q2 indicator does not predict a specific error rate, but rather indicates that a specific final portion of the read should not be used in further analyses.* Also, the quality score encoded as “B” letter may occur internally within reads at least as late as pipeline version 1.6, as shown in the following example:

```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACCTTNNNNNNNNNNTAGTTTCTTGAGATTTGTTGGGGGAGACATTTTTGTGATTGCCTT
+HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
efcfffffcfeefffcffffffddf`feed]`]_Ba_^__[YBBBBBBBBBBRTT\]][]dddd`ddd^dddadd^BBBBBBBBBBBBBBBBBBBBB
```

An alternative interpretation of this ASCII encoding has been proposed. Also, in Illumina runs using PhiX controls, the character ‘B’ was observed to represent an “unknown quality score”. The error rate of ‘B’ reads was roughly 3 phred scores lower the mean observed score of a given run.

- Starting in Illumina 1.8, the quality scores have basically returned to the use of the Sanger format (Phred+33).

1.3 File extension

There is no standard file extension for a FASTQ file, but .fq and .fastq, are commonly used.

1.4 See also

- :ref:`fasta`

1.5 References

.. [1] Cock et al (2009) The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Research,

.. [2] Illumina Quality Scores, Tobias Mann, Bioinformatics, San Diego, Illumina 1___

.. |Relationship between Q and p using the Sanger (red) and Solexa (black) equations (described above). The vertical dotted line indicates $p = 0.05$, or equivalently, Q Å 13.| image:: Probability metrics.png

See http://en.wikipedia.org/wiki/FASTQ_format

Chapter 2

The *OBITools* commands

2.1 Specifying the input files to *OBITools* commands

2.2 Options common to most of the *OBITools* commands

2.2.1 Specifying input format

Five sequence formats are accepted for input files. Fasta and Fastq are the main ones, EMBL and Genbank allow the use of flat files produced by these two international databases. The last one, ecoPCR, is maintained for compatibility with previous *OBITools* and allows to read *ecoPCR* outputs as sequence files.

- `--ecopcr` : Read data following the *ecoPCR* output format.
- `--embl` Read data following the *EMBL* flatfile format.
- `--genbank` Read data following the *Genbank* flatfile format.

Several encoding schemes have been proposed for quality scores in Fastq format. Currently, *OBITools* considers Sanger encoding as the standard. For reasons of compatibility with older datasets produced with *Solexa* sequencers, it is possible, by using the following option, to force the use of the corresponding quality encoding scheme when reading these older files.

- `--solexa` Decodes quality string according to the Solexa specification. (default: false)

2.2.2 Specifying output format

Only two output sequence formats are supported by OBITools, Fasta and Fastq. Fastq is used when output sequences are associated with quality information.

Otherwise, Fasta is the default format. However, it is possible to force the output format by using one of the following two options. Forcing the use of Fasta results in the loss of quality information. Conversely, when the Fastq format is forced with sequences that have no quality data, dummy qualities set to 40 for each nucleotide are added.

- `--fasta-output` Read data following the ecoPCR output format.
- `--fastq-output` Read data following the EMBL flatfile format.

OBITools allows multiple input files to be specified for a single command.

- `--no-order` When several input files are provided, indicates that there is no order among them. (default: false)

2.2.3 Format of the annotations in Fasta and Fastq files

OBITools extend the Fasta and Fastq formats by introducing a format for the title lines of these formats allowing to annotate every sequence. While the previous version of OBITools used an *ad-hoc* format for these annotation, this new version introduce the usage of the standard JSON format to store them.

On input, OBITools automatically recognize the format of the annotations, but two options allows to force the parsing following one of them. You should normally not need to use these options.

- `--input-OBI-header` FASTA/FASTQ title line annotations follow OBI format. (default: false)

- `--input-json-header` FASTA/FASTQ title line annotations follow json format. (default: false)

On output, by default annotation are formatted using the new JSON format. For compatibility with previous version of OBITools and with external scripts and software, it is possible to force the usage of the previous OBITools format.

- `--output-OBI-header|-O` output FASTA/FASTQ title line annotations follow OBI format. (default: false)

- `--output-json-header` output FASTA/FASTQ title line annotations follow json format. (default: false)

2.2.3.1 System related options

- `--debug` (default: false)
- `--help\|-h\|-?` (default: false)
- `--max-cpu <int>` Number of parallele threads computing the result (default: 10)
- `--workers\|-w <int>` Number of parallele threads computing the result (default: 9)

2.3 Metabarcode design and quality assessment

2.3.0.1 `obipcr`

Replace the `ecoPCR` original *OBITools*

2.4 File format conversions

2.4.0.1 `obiconvert`

2.5 Sequence annotations

2.5.0.1 `obitag`

2.6 Computations on sequences

2.6.1 `obipairing`

Replace the `illuminapairedends` original *OBITools*

2.6.1.1 `obimultiplex`

Replace the `ngsfilter` original *OBITools*

2.6.1.2 `obicomplement`

2.6.1.3 `obiclean`

2.6.1.4 `obiuniq`

2.7 Sequence sampling and filtering

2.7.0.1 `obigrep`

2.7.1 Utilities

2.7.1.1 `obicount`

2.7.1.2 `obidistribute`

2.7.1.3 `obifind`

Replace the `ecofind` original *OBITools.*

Chapter 3

Reference documentation for the GO *OBITools* library

3.1 BioSequence

The `BioSequence` class is used to represent biological sequences. It allows for storing : - the sequence itself as a `[]byte` - the sequencing quality score as a `[]byte` if needed - an identifier as a `string` - a definition as a `string` - a set of *(key, value)* pairs in a `map[sting]interface{}`

BioSequence is defined in the obiseq module and is included using the code

```go
import (
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)
```

3.1.1 Creating new instances

To create new instance, use

- `MakeBioSequence(id string, sequence []byte, definition string) obiseq.BioSequence`
- `NewBioSequence(id string, sequence []byte, definition string) *obiseq.BioSequence`

Both create a `BioSequence` instance, but when the first one returns the instance, the second returns a pointer on the new instance. Two other functions `MakeEmptyBioSequence`, and `NewEmptyBioSequence` do the same job but provide an uninitialized objects.

- `id` parameters corresponds to the unique identifier of the sequence. It mist be a string constituted of a single word (not containing any space).

- `sequence` is the DNA sequence itself, provided as a `byte` array (`[]byte`).
- `definition` is a `string`, potentially empty, but usualy containing a sentence explaining what is that sequence.

```
import (
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)

func main() {
    myseq := obiseq.NewBiosequence(
        "seq_GH0001",
        bytes.FromString("ACGTGTCAGTCG"),
        "A short test sequence",
        )
}
```

When formated as fasta the parameters correspond to the following schema

```
>id definition containing potentially several words
sequence
```

3.1.2 End of life of a `BioSequence` instance

When a `BioSequence` instance is no more used, it is normally taken in charge by the GO garbage collector. You can if you want call the `Recycle` method on the instance to store the allocated memory element in a `pool` to limit allocation effort when many sequences are manipulated.

3.1.3 Accessing to the elements of a sequence

The different elements of an `obiseq.BioSequence` must be accessed using a set of methods. For the three main elements provided during the creation of a new instance methodes are :

- `Id() string`
- `Sequence() []byte`
- `Definition() string`

It exists pending method to change the value of these elements

- `SetId(id string)`
- `SetSequence(sequence []byte)`
- `SetDefinition(definition string)`

```
import (
    "fmt"
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)
```

```go
func main() {
    myseq := obiseq.NewBiosequence(
        "seq_GH0001",
        bytes.FromString("ACGTGTCAGTCG"),
        "A short test sequence",
        )

    fmt.Println(myseq.Id())
    myseq.SetId("SPE01_0001")
    fmt.Println(myseq.Id())
}
```

3.1.3.1 Different ways for accessing an editing the sequence

If `Sequence()`and `SetSequence(sequence []byte)` methods are the basic ones, several other methods exist.

- `String() string` return the sequence directly converted to a `string` instance.
- The `Write` method family allows for extending an existing sequence following the buffer protocol.
  - `Write(data []byte) (int, error)` allows for appending a byte array on 3' end of the sequence.
  - `WriteString(data string) (int, error)` allows for appending a `string`.
  - `WriteByte(data byte) error` allows for appending a single `byte`.

The `Clear` method empties the sequence buffer.

```go
import (
    "fmt"
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)

func main() {
    myseq := obiseq.NewEmptyBiosequence()

    myseq.WriteString("accc")
    myseq.WriteByte(byte('c'))
    fmt.Println(myseq.String())
}
```

3.1.3.2 Sequence quality scores

Sequence quality scores cannot be initialized at the time of instance creation. You must use dedicated methods to add quality scores to a sequence.

To be coherent the length of both the DNA sequence and que quality score sequence must be equal. But assessment of this constraint is realized. It is of the programmer responsability to check that invariant.

While accessing to the quality scores relies on the method `Quality() []byte`, setting the quality need to call one of the following method. They run similarly to their sequence dedicated conterpart.

- `SetQualities(qualities Quality)`
- `WriteQualities(data []byte) (int, error)`
- `WriteByteQualities(data byte) error`

In a way analogous to the `Clear` method, `ClearQualities()` empties the sequence of quality scores.

Chapter 4

Annexes

4.0.1 Sequence attributes

4.0.1.1 Reserved sequence attributes

4.0.1.1.1 `ali_dir`

4.0.1.1.1.1 Type : `string` The attribute can contain 2 string values `"left"` or `"right"`.

4.0.1.1.1.2 Set by the *obipairing* tool The alignment generated by *obipairing* is a 3'-end gap free algorithm. Two cases can occur when aligning the forward and reverse reads. If the barcode is long enough, both the reads overlap only on their 3' ends. In such case, the alignment direction `ali_dir` is set to *left*. If the barcode is shorter than the read length, the paired reads overlap by their 5' ends, and the complete barcode is sequenced by both the reads. In that later case, `ali_dir` is set to *right*.

4.0.1.1.2 `ali_length`

4.0.1.1.2.1 Set by the *obipairing* tool Length of the aligned parts when merging forward and reverse reads

4.0.1.1.3 `count` : the number of sequence occurrences

4.0.1.1.3.1 Set by the *obiuniq* tool The `count` attribute indicates howmany strictly identical sequences have been merged in a single record. It contains an integer value. If it is absent this means that the sequence record represents a single occurrence of the sequence.

4.0.1.1.3.2 Getter : method `Count()` The `Count()` method allows to access to the count attribute as an integer value. If the `count` attribute is not defined for the given sequence, the value *1* is returned

4.0.1.1.4 `merged_*`

4.0.1.1.4.1 Type : `map[string]int`

4.0.1.1.4.2 Set by the *obiuniq* tool The `-m` option of the *obiuniq* tools allows for keeping track of the distribution of the values stored in given attribute of interest. Often this option is used to summarise distribution of a sequence variant accross samples when *obiuniq* is run after running *obimultiplex.* The actual name of the attribute depends on the name of the monitored attribute. If `-m` option is used with the attribute *sample*, then this attribute names *merged_sample.*

4.0.1.1.5 `mode`

4.0.1.1.5.1 Set by the *obipairing* tool `obitag_ref_index`

4.0.1.1.5.2 Set by the *obirefidx* tool. It resumes to which taxonomic annotation a match to that sequence must lead according to the number of differences existing between the query sequence and the reference sequence having that tag.

4.0.1.1.5.3 Getter : method `Count()`

4.0.1.1.6 `pairing_mismatches`

4.0.1.1.6.1 Set by the *obipairing* tool

4.0.1.1.7 `score`

4.0.1.1.7.1 Set by the *obipairing* tool

4.0.1.1.8 `score_norm`

4.0.1.1.8.1 Set by the *obipairing* tool