BiOCamLib

BiOCamLib is the OCaml foundation upon which a number of the bioinformatics tools I developed are built.

It mostly consists of a library — you'll need to clone this repository if you want to manually compile other programs I've developed, notably SiNPle or KPop. You might also use the library for your own programs, if you are familiar with OCaml and patient enough to read the code.

As a bonus, BiOCamLib comes bundled with a few programs:

- `RC`, which can efficiently compute the reverse complement of (possibly very long) sequences. Each sequence should be input on a separate line — lines are processed one by one and not buffered. I use this program in many of my workflows.
- `Octopus`, which is a high-throughput program to compute the transitive closure of strings. This is useful to cluster things.
- `Parallel`, which allows you to split and process an input file chunk-wise using the reader/workers/writer model implemented in `BiOCamLib.Tools.Parallel`. You can see it as a demonstration of the capabilities of the library, but I also often use it as a useful tool to solve real-life problems.
- `FASTools`, which is a Swiss-knife tool for the manipulation of FASTA/FASTQ files. It supports all formats (FASTA, single- and paired-end FASTQ, interleaved FASTQ) and a simpler tabular format whereby FASTA/FASTQ records are represented as tab-separated lines. It facilitates format interconversions and other manipulations.

Installing `RC`, `Octopus`, `Parallel`, and `FASTools`

> ⚠️ Note that the only operating systems we officially support are Linux and MacOS. ⚠️
>
> OCaml is highly portable and you might be able to manually compile/install everything successfully on other platforms (for instance, Windows) but you will have to do it yourself.

There are several possible ways of installing the software on your machine: through `conda`; by downloading pre-compiled binaries (Linux and MacOS x86_64 only); or manually.

Conda channel

> 🚧 Coming soon! 🚧

Pre-compiled binaries

You can download pre-compiled binaries for Linux and MacOS x86_64 from our releases.

Manual install

Alternatively, you can install `RC`, `Octopus`, `Parallel`, and `FASTools` manually by cloning and compiling the BiOCamLib sources. You'll need an up-to-date distribution of the OCaml compiler and the Dune package manager for that. Both can be installed through OPAM, the official OCaml distribution system. Once you have a working OPAM distribution you'll also have a working OCaml compiler, and Dune can be installed with the command

```
$ opam install dune
```

if it is not already present. Make sure that you install OCaml version 4.12 or later.

Then go to the directory into which you have downloaded the latest BiOCamLib sources, and type

```
$ ./BUILD
```

That should generate the executables `RC`, `Octopus`, `Parallel`, and `FASTools`. Copy them to some favourite location in your PATH, for instance `~/.local/bin`.

Using `RC`

`RC` inputs sequences from standard input and outputs their reverse complement to standard output, one sequence at the time. Hence, `RC` can be conveniently used in a subprocess. For example, the command

```
$ echo GAtTaCA | RC
```

would produce `TGtAaTC`. Note that non-`[ACGTacgt]` characters are output unmodified, so sequence validation and linting must be performed elsewhere whenever they are necessary.

Command line options

This is the full list of command line options available for the program `RC`. You can visualise the list by typing

```
$ RC -h
```

in your terminal. You will see a header containing information about the version:

```
This is the RC program (version 0.2)
 (c) 2023 Paolo Ribeca, <paolo.ribeca@gmail.com>
```

followed by detailed information. The general form(s) the command can be used is:

```
RC [OPTIONS]
```

Algorithm

Option	Argument(s)	Effect	Note(s)
`-C` `--no-complement`		do not base-complement the sequence	default=*base-complement*

Miscellaneous

Option	Argument(s)	Effect	Note(s)
`-V` `--version`		print version and exit	
`-h` `--help`		print syntax and exit	

Using `Octopus`

`Octopus` reads from its standard input equivalence relations, one set of relations per line. Each line consists of a set of strings separated by whitespace; if any two labels appear on the same line, they are considered to belong to the same equivalence class. When all the input has been parsed, `Octopus` outputs all the labels seen in the input sorted according to their equivalence class — each line contains one equivalence class, with its member string labels separated by a `\t` character. The order in which classes appear is kept, but elements within the class will be lexicographically sorted. For example, the command

```
$ (cat <<___
A duh
  b  C
c f e
 duh zz x
b c
___
) | Octopus
```

(without the first `$` prompt character) will result in the output

```
A       duh     x       zz
C       b       c       e       f
```

(tab-separated).

Command line options for `Octopus`

This is the full list of command line options available for the program `Octopus`. You can visualise the list by typing

```
$ Octopus -h
```

in your terminal. You will see a header containing information about the version:

```
This is the Octopus program (version 0.4)
 (c) 2016-2023 Paolo Ribeca, <paolo.ribeca@gmail.com>
```

followed by detailed information. The general form(s) the command can be used is:

```
Octopus [OPTIONS]
```

Miscellaneous

Option	Argument(s)	Effect	Note(s)
`-V` `--version`		print version and exit	
`-h` `--help`		print syntax and exit	

Command line options for `Parallel`

This is the full list of command line options available for the program `Parallel`. You can visualise the list by typing

```
$ Parallel -h
```

in your terminal. You will see a header containing information about the version:

```
This is the Parallel program (version 0.4)
 (c) 2019-2022 Paolo Ribeca, <paolo.ribeca@gmail.com>
```

followed by detailed information. The general form(s) the command can be used is:

```
Parallel [OPTIONS] -- [COMMAND TO PARALLELIZE AND ITS OPTIONS]
```

Command to parallelize

Option	Argument(s)	Effect	Note(s)
`--`		consider all the subsequent parameters as the command to be executed in parallel. At least one command must be specified	*(mandatory)*

Input/Output

Option	Argument(s)	Effect	Note(s)
`-l` `--lines-per-block`	*<positive_integer>*	number of lines to be processed per block	default=*10000*
`-i` `--input`	*<input_file>*	name of input file	default=*stdin*
`-o` `--output`	*<output_file>*	name of output file	default=*stdout*

Miscellaneous

Option	Argument(s)	Effect	Note(s)
`-t` `--threads`	*<positive_integer>*	number of concurrent computing threads to be spawned (default automatically detected from your configuration)	default=*nproc*
`-v` `--verbose`		set verbose execution	default=*false*
`-d` `--debug`		output debugging information	default=*false*
`-h` `--help`		print syntax and exit	

Command line options for `FASTools`

This is the full list of command line options available for the program `FASTools`. You can visualise the list by typing

```
$ FASTools -h
```

in your terminal. You will see a header containing information about the version:

```
This is the FASTools program (version 0.5)
 (c) 2022-2023 Paolo Ribeca, <paolo.ribeca@gmail.com>
```

followed by detailed information. The general form(s) the command can be used is:

```
FASTools [OPTIONS]
```

Working mode. Executed delayed in order of specification, default=*compact*.

Option	Argument(s)	Effect	Note(s)
`compact` `-c` `--compact`		put each FASTA/FASTQ record on one tab-separated line	
`expand` `-e` `--expand`		split each tab-separated line into one or more FASTA/FASTQ records	
`match` `-m` `--match`	*<regexp>*	select matching sequence names in FASTA/FASTQ records or tab-separated lines. For paired-end files, the pair matches when at least one name matches	
`revcom` `-r` `--revcom`		reverse-complement sequences in FASTA/FASTQ records or tab-separated lines	
`dropq` `-d` `--dropq`		drop qualities in FASTA/FASTQ records or tab-separated lines	

Input/Output. Executed delayed in order of specification, default=*-F*.

Option	Argument(s)	Effect	Note(s)
`-f` `--fasta`	*<fasta_file_name>*	process FASTA input file containing sequences	
`-F`		process FASTA sequences from standard input	
`-s` `--single-end`	*<fastq_file_name>*	process FASTQ input file containing single-end sequencing reads	
`-S`		process single-end FASTQ sequencing reads from standard input	
`-p` `--paired-end`	*<fastq_file_name1>* *<fastq_file_name2>*	process FASTQ input files containing paired-end sequencing reads	
`-P`		process interleaved FASTQ sequencing reads from standard input	
`-t` `--tabular`	*<tabular_file_name>*	process input file containing FAST[A|Q] records as tab-separated lines	
`-T`		process FAST[A|Q] records in tabular form from standard input	
`-l` `--linter`	*'none'|'DNA'|'dna'|'protein'*	sets linter for sequence. All non-base (for DNA) or non-AA (for protein) characters are converted to unknowns	default=*none*
`--linter-keep-lowercase`	*<bool>*	sets whether the linter should keep lowercase DNA/protein characters appearing in sequences rather than capitalise them	default=*false*
`--linter-keep-dashes`	*<bool>*	sets whether the linter should keep dashes appearing in sequences or convert them to unknowns	default=*false*
`-o` `--output`	*<output_file_name>*	set the name of the output file. Files are kept open, and it is possible to switch between them by repeatedly using this option. Use '/dev/stdout' for standard output	default=*/dev/stdout*
`-O` `--paired-end-output`	*<output_file_name_1>* *<output_file_name_2>*	set the names of paired-end FASTQ output files. Files are kept open, and it is possible to switch between them by repeatedly using this option. Use '/dev/stdout' for standard output	default=*/dev/stdout* * /dev/stdout*
`--flush` `--flush-output`		flush output after each record (global option)	default=*do not flush*

Miscellaneous

Option	Argument(s)	Effect	Note(s)
`-v` `--verbose`		set verbose execution (global option)	default=*false*
`-h` `--help`		print syntax and exit	