FilterReads

FilterReads is a fast kMer-based filter that pulls out matching reads from WGS datasets (or contigs from sets of contigs). Each read is tiled into kMers, and these are then matched against a set of kMer filters. Reads with sufficient numbers of matching kMers are kept (or discarded). The kMer filters are simply sets of kMers tiled from a set of reference sequences, and are generated by BuildFilter. Some pre-built filters are available via the GHAP distribution (https://doi.org/10.4225/08/59f98560eba25 and https://cloudstor.aarnet.edu.au/plus/s/pUvwME6fMeCcXzL).

Any number of filters can be specified, and they can be inclusive or exclusive. Reads that pass through FilterReads must have enough kMer matches onto any one of the specified inclusive filters, and not have too many matches on any of the exclusive filters. For example, one of the examples below was written to filter bacterial 16S sequences from mixed human gut wall samples. These samples were a mixture of human and microbial RNA/DNA and the filter pulled out the bacterial 16S while discarding the very similar human 18S and mitochondrial 16S sequences. The examples also show that FilterReads can be used on contigs as well as reads.

FilterReads is a command-line program with the following cryptic usage hint:

```
FilterReads -r tag -t threads [-pairs] [-matches] [-full] [-len minLen] [-s] [-qt
minQual] [-o outputDir] [-fasta] [-discards] [+f|fz includeFilter
minMatches[%|pct]] [-f|fz excludeFilter minMatches[%|pct]] readsFNs
```

A typical run of FilterReads is…

```
FilterReads -r 16S -t 8 -full -qt 30 +fz RefSeq16S_25.mer 50% W1_?.fastq
```

…which will filter whatever files match `W1_?.fastq` for reads where at least 50% of the 25-mers are found in `RefSeq16S_25.mer` (a 25-mer filter built from RefSeq16S). Matches are required at both ends of the read (-full), and the 25-mers are allowed to have single-base differences. The filtered reads will be written to something like `W1_1_16S.fastq`.

More complex examples are:

```
FilterReads -r 16S -t 8 +f current_prokMSA_unaligned_May2011_25.mer 10%
-f Human_mitochondria_25.mer 20% -f human_18S_25.mer 20% GH_02d_ATCACG_R?.fastq
```

```
FilterReads -r aflatoxin -t 16 +f aflatoxinGenes_25.mer 1000 Contigs\F36-HGM-
57\contigs_F36-HGM-57-11.fa
```

The full set of FilterReads parameters is:

`-r tag`
`-run tag`
	This tag is added to the end of each to-be-filtered file name to create the name of the file that will hold the filtered reads.
	e.g. `-r 16S W1_1.fastq` → `W1_1_16S.fastq`

`-t number_of_threads`
`-threads number_of_threads`
	Number of parallel filtering threads. Default is 1.

`-pairs`	Keeps read-pairing across pairs of files. Pairs of files (typically R1 & R2) are processed together, and both reads from each pair must pass the filtering test before they can be written to their respective 'filtered' files. If either read in a pair doesn't get enough inclusive (+f) matches, or gets too many exclusive matches (-f), both the reads will be dropped. Setting this option has a performance implication as FilterReads will be

	reading and writing multiple files at once, and this will result in slower IO on old-fashioned hard disk drives. Default is to filter all separately, and not maintain read pairing.
`-fasta`	Forces the filtered reads to be written in FASTA format. Default is to write out the filtered reads in the same format as used for the unfiltered reads.

`-qt minimum_qual_value`
`-qualTrim minimum_qual_value`
	Tail-trims each read prior to filtering. Uses a sliding window to cut back the tails of reads until the specified quality level is reached. Default is no trimming.

`-l minimum_read_length`
`-length minimum_read_length`
	Minimum read length (after qual-trimming). Reads short than this length will be dropped.

`-full` `-fullLength`	Matching kMers must appear in both halves of the reads. Improves overall filtering accuracy at the expense of reads covering the end of the target region. Default is to say that reads can be accepted with enough matches at just one end.

`+f includeFilter minMatches[%|pct]`
 	
`+fz includeFilter minMatches[%|pct]`
	Inclusive filter. Reads must have the specified number of matches onto the kMers in this filter. The required number of matches can be a fixed number, such as 100, or a percentage of the read length, such as 50%. The % character is a bit special in some scripting languages so you can use ‘pct’, as in 50pct.
	
	Matches can be exact (+f) or single-base mismatches can be allowed (+fz).
	
	Any number of these filters can be specified, and a successful match on any one of them is enough to declare that the read is matching.

`-f excludeFilter minMatches[%|pct]`
 	
`-fz excludeFilter minMatches[%|pct]`
	Exclusive filter. Reads must not have more than the specified number of matches onto this filter.
	
	Any number of these filters can be specified, and a match on any of them is enough to declare that the read should be rejected.

`-s`	Search for matching file names in any subdirectories as well as in the current directory. Default is to only look for matching file names in the current directory.
`-m` `-matches`	Writes out an alignment of matching kMers and reads. This option should only be used on small datasets.

`-discards`	Save any reads that were dropped because they matched one of the exclusive filters. These reads are written to '_discards' files. e.g. S1_270_reads_anonymous_R?_discards.fasta
`readsFNP`	List of file names/patterns of reads to be filtered. e.g. S1_270_reads_anonymous_R?.fasta

FilterReads is written in C# and is provided pre-compiled for Windows, OSX and common Linux variants. These pre-compiled code files should be stand-alone and should not require the installation of any additional run-time libraries.

You can compile FilterReads yourself using Mono (under OSX and Linux), or under Visual Studio on Windows. You'll need the `mono-devel` package installed on Linux or OSX.

The FilterReads code itself is in Program.cs in this directory, and you'll also need to download kMers.cs, SeqFiles.cs and Sequence.cs from WorkingDocsCoreLibrary. One simple way of compiling Kelpie (if you need to) is to make a directory containing these 4 .cs files and then run `'csc /out:FilterReads.exe /target:exe *.cs'` from this directory. Earlier versions of mono use 'msc' rather than 'csc' but the syntax of the compilation command is the same. The C# compiler will produce the executable `FilterReads.exe`. This code file can be run natively on Windows or via `mono FilterReads` on other platforms.

The native Linux and OSX executables provided with the package were produced by cross-compiling `FilterReads.exe` with `mkbundle`, targeting various Linux and OSX releases.

```
mkbundle -o ubuntu-18.04\ FilterReads FilterReads.exe --simple --cross mono-5.12.0-
ubuntu-18.04-x64
```

You can also use `mkbundle` to generate a native executable for your current system. You'll need to know where the `mono-devel` installation put the `machine.config` and the `mono` libraries.

```
mkbundle -o FilterReads --simple FilterReads.exe --machine-config
/etc/mono/4.5/machine.config -L /apps/mono/5.4.1.7/lib/mono/4.5
```

The mkbundle command has been changing with recent mono releases, and you may need to try other variants, such as:

```
mkbundle -o FilterReads --cross default FilterReads.exe
```

It is expected that all use of mono will be replaced late in 2018 once .NET Core 3 arrives, with direct multi-platform code generation.