SNAP User Manual

By William J. Bolosky and Arun Subramaniyan

Version of October 2021 corresponding to SNAP version 1.1

Contents

Introduction

The Scalable Nucleotide Alignment Program (SNAP) is a program for aligning and processing short DNA reads against a known reference. It includes support for multiple input and output formats, as well as the ability to sort, index and mark duplicates without having to use external tools. SNAP runs on both Windows and Linux systems. This manual describes how to use SNAP and its various features.

SNAP's workflow is in two stages. First it must build an index. This takes a reference genome (in FASTA format) and constructs a data structure that allows SNAP to very quickly map *k*-mers from a read to the set of locations in the reference that are that k-mer. You need only do this once per reference and value of *k*. The section on indexing describes the tradeoffs in selecting *k*.

Once there is an index, SNAP can align reads against the reference represented in that index. It has two modes: single-end and paired-end. After aligning, SNAP can optionally sort and mark duplicates in a separate, post-alignment pass (but a single run of the program).

SNAP also has special modes for different uses. One common application of SNAP is to use it in a metagenomics pipeline to filter out reads that aren't interesting. For example, when trying to extract bacterial reads from a culture, it is helpful to first eliminate the human reads. Nearly all the effort in running an alignment is finding the best alignment among many candidates. When it's only interesting to find that there is a good alignment regardless of what that alignment is, it's possible to stop much more quickly. In addition, you can configure SNAP only to emit unaligned reads (which is useful in the metagenomics filtering context), aligned reads, or reads that had alignments with a high enough mapping quality.

By default SNAP will only emit what it thinks is the best alignment for a read. There are command line options to have it emit multiple possible alignments in for some reads, which can be configured in various ways to determine which and how many alignments should be emitted. For example, there is a mode designed for metagenomics applications that will emit at most one alignment per contig even if there are multiple good-enough alignments for a given contig.

Index Building

SNAP's index is a set of hash tables that map fixed-length sequences of bases ("*k*-mers") to the set of locations in the reference that have exactly that sequence, as well as a copy of the reference. SNAP indices tend to be several times the size of the reference, though there is considerable variation based on the parameters used to build the index and to some extent the level of repetitiveness of the reference. SNAP indices are stored in a directory that contains four files, but it's easiest to just treat it as a single unit.

To build an index, you need to run the `snap`[1] `index` command.

The simplest version is:

`snap index` <input.fasta> <output directory>[2]

An index build runs in several phases. First it loads the reference, then it writes it out in SNAP's internal format. Then it computes the bias table (explained below), allocates memory for the hash tables, and computes the hash tables. Finally, it computes the overflow table and writes out the hash and overflow tables. Most of the time is spent computing the hash and overflow tables. SNAP is automatically parallel in building its index and will use one thread per core on the machine unless told to do otherwise.

There are several parameters that affect index building. By far the most important one is the seed size, specified with the –s parameter:

`snap index` <input.fasta> <output directory> {`-s` <seed size>}[3]

The seed size determines the number of consecutive bases that are mapped by the snap index to a set of locations in the genome, *i.e. k*. It can range from 8 to 32 with a default value of 27. Smaller seed sizes allow SNAP to see places in the genome that don't match as well with the read being aligned, so in principle would improve alignment quality. However, there will be more locations that match shorter seeds, so a smaller seed size increases the amount of work done and reduces alignment speed. We have measured the accuracy of variant calls based on SNAPs alignments of genomes with known truth sets and found that reducing seed size too much can increase the error rate due to too many false positives. Our intuition that smaller seeds would produce better results at the cost of decreased performance was consistently disproved by the data. There is some tradeoff between recall and precision, with larger seed sizes favoring precision over recall. It may also make sense to use shorter seeds with reads that are shorter than 100 bases, though we do not have data for this case.

Commented [BB1]: Update this when we decide the 1.1 value

[1] The name of the executable is different on Windows and Linux. It's "snap.exe" on Windows and "snap-aligner" on Linux (to avoid conflicting with an unrelated tool). In this manual we just use "`snap`" (in lower case and fixed-width font) for both.

[2] In this manual, literal strings in command lines are represented in a `fixed-width font`, while values that you need to supply are enclosed in angle brackets and printed in a variable-width font.

[3] Curly braces surround optional parts of a command line and are not literally to be typed. A superscript asterisk after curly braces means that the argument may be specified any number of times, subject to the operating system's command line length limit.

The key size and exact parameters

The contents of the hash table are key values and locations. In principle, a key is just the seed that's being indexed. Each base in the seed is represented in two bits of key, so a seed size of 16 would result in keys of 4 bytes (= 16 bases * 2 bits/base / 8 bits/byte) and a seed size of 32 bases would result in keys of 8 bytes. However, SNAP employs an optimization to reduce the hash table size. Instead of building a single hash table for the entire index, it separates the seeds into a prefix and a body. There is one hash table for each possible prefix and only the body part of the key is stored in the hash tables. So, for example, if the seed size is 17 bases and the prefix is one base, then there will be four hash tables, one for each of the four possible prefixes: T, C, G and A. The remaining 16 bases would be encoded in four bytes in each of the hash table entries. This saves having to store the entire key in each hash table row.

SNAP does not index portions of the reference that contain anything other than the four bases, for example N.

Controlling the size of the prefix is done implicitly. SNAP lets you specify the size of the seed and the number of bytes of key in the hash table entries, from which it computes the prefix size and then the number of hash tables. The number of hash tables is $4^{seed\ size - 4*key\ size}$ where seed size is in bases and key size is in bytes. Key size must be between 2 and 8 bytes and the number of hash tables must be between 1 and $4^8 = 65{,}536$. This puts a constraint on the valid set of key sizes for a given seed size. If you exceed this bound, SNAP will refuse to build the index and give a helpful error message suggesting that you change either key size or seed size.

In most references not all possible seeds are equally represented, and hence not all possible seed prefixes are equiprobable. This means that the hash tables will not all have the same number of entries in them. Because SNAP uses fixed-size hash tables, before building them it needs to know (approximately) how many unique seeds will be in each. It determines this by making a sampling pass over the reference before it decides the hash table sizes. This is what's happening in the "bias computation" phase of index building. You may tell SNAP to get the hash table sizes exactly right rather than estimating (at the cost of slowing down index build) by specifying the -exact flag. This may result in a somewhat smaller index. If, for some reason, the estimation is off by enough that a hash table overflows and the index build fails, -exact will correct that (we've never seen this happen, though).

You specify the key size to SNAP with the `-keysize` flag and tell it to compute exact hash table sizes with the `-exact` flag:

`snap index` <input.fasta> <output directory> {`-keysize` <key size in bytes>} {-exact}

SNAP uses case-insensitive matching for the `-keysize` flag, so for example `-KeySize` or `-keySize` also work[4].

If you do not specify a key size, SNAP computes one based on the seed size that results in between 16 and 1024 hash tables (except for seed sizes 8 and 9, which have 1 and 4 hash tables respectively).

The location size parameter

The other part of the content of the hash table entries is the location. This is what SNAP uses to represent the set of locations to which a seed maps. For most references and most seed sizes most

[4] We got tired of forgetting the correct capitalization.

seeds that occur in the reference occur exactly once. (And a few occur a very large number of times.) Rather than having most of the hash table entries point at a singleton set and then incur an extra cache miss with indirection, SNAP optimizes the exactly-once case by just storing the genome location directly in the hash table entry. For seeds with more than one occurrence in the genome it stores an index into a table of sets of genome locations (the *overflow table*). The size of the portion of the hash table entry used to store these values is called the "location size." Because every genome location must be representable, there is a lower bound on the location size:

$$location\ size\ \geq \left\lceil \frac{log_2(reference\ genome\ size)}{8} \right\rceil$$

However, there must be additional space to store indices into the overflow table. This depends on the repetitiveness of the reference and the seed size. For instance, for hg38 location size must be at least 5 for seeds smaller than 20 bases, even though 4 bytes is sufficient for the location of the reference itself. If you specify too small of a location size the index build will fail with an appropriate error message. If you select too large it will succeed, but the index will be larger than it needs to be, and performance may be hurt somewhat by reducing the effectiveness of the processor cache (as well as increasing index load time).

The default location size is chosen to work with the human genome reference. If seed size is 20 or larger it is 4, if seed size is smaller than 20 it is 5. If you are using a reference that is much smaller than the human genome you might want to use a smaller location size in order to save space.

You specify the location size with the `-locationSize` flag:

`snap index` <input.fasta> <output directory> {`-locationSize` <location size in bytes>}

SNAP uses case-insensitive matching for the `-locationSize` flag.

Large hash tables

It is common to want to find the set of genome locations of a seed and of its reverse complement. Doing that requires two lookups in the SNAP index. SNAP has an index build option that stores them both together. For these kinds of indices the hash table rows have one key and two locations: one for the key and one for its reverse complement.

It's common for a reference genome not to have an instance of the reverse complement of most of the seeds in the genome, which means that very often an index that stores both together will have more unused space occupied by empty location fields. So, storing both the seed and its reverse complement together trades index size for speed. For example, for the human hg38 reference with a seed size of 20, storing both complements in the hash table increases the size by about a third (from just over 30GB to just over 40GB), but with the large index alignment runs 10% faster (your mileage will vary).

`snap index` <input.fasta> <output directory> {`-large`}

Specifying contig names

FASTA breaks a reference genome into contigs. These are regions that mappings can't span and that may represent different DNA molecules or contiguous pieces of an assembly. In the FASTA file each contig starts with a line containing its name and possibly other information. There is no uniform

standard to determine what is the contig name and what is the other information. SNAP lets you specify at index build time a set of characters that will terminate a contig name. These characters are specified using one or both of the -B and -bSpace- flags. -B specifies all the characters that terminate a contig name other than space and tab. They must be specified in a single -B flag. So, for example -B|,_ would say that when a bar "|", comma "," or underscore "_" occurs in a contig name in a FASTA that the actual contig name ends at the first character before the first one of these.

`-bSpace-` indicates that the space and tab characters are not contig name terminators. By default they are. There is a `-bSpace` flag (without the trailing dash) that says that space and tab characters are name terminators, but it does nothing as that is the default. It exists for backward compatibility from when the default was different.

`snap index` <input.fasta> <output directory> {`-B`<characters>} {`-bSpace|-bSpace-`}

By default, the entire string in the contig name line of the FASTA file is used as the contig name for SNAP.

Specifying ALT contigs

Some references, notably the Human Genome Reference Consortium references starting at version 38 (GRCh38 or hg38), contain ALT contigs. An ALT contig is a different version of a section of the reference that's found in some fraction of the population. Often, there is quite a bit of overlap between ALT contigs and the corresponding DNA in the reference genome. If an aligner is unaware of ALT contigs, it may map reads to the ALT rather than to the primary assembly when they're close, and even if it doesn't, its estimation of mapping quality may be reduced. SNAP has support for understanding ALT contigs. This is explained in detail in the section of the manual that describes mapping. This section explains the various ways to specify which contigs are ALT when building an index.

If you specify no flags, then any contig with a name ending in "_alt" or beginning with "HLA-" (regardless of capitalization for either) will be marked as an ALT contig. To disable this behavior, specify the `-AutoAlt-` flag (the trailing dash means "off"):

There are other ways to specify ALT contigs. `-maxAltContigSize` causes SNAP to mark contigs of that size or less as ALT unless overridden using one of the non-ALT options described below.

`-altContigName` specifies the name of a contig that's to be marked ALT. `-altContigFile` specifies a file that contains a list of contigs to be marked ALT, one per line. These arguments may be specified as many times as needed. The specified contig names are not case sensitive.

`-nonAltContigName` and `-nonAltContigFile` do the same thing, except that they specify contigs that are never to be marked ALT. They also may be specified as many times as needed. The specified contig names are not case sensitive. Specifying a contig as non-ALT overrides any other determination of being ALT: by size, by being explicitly listed, or, by ending in "_alt."

`snap index` <input.fasta> <output directory> {`-AutoAlt-`} {`-altContigName` <contig to mark>}* {`-altContigFile` <file with names of contigs>}* {`-nonAltContigName` <contig to mark>}* {`-nonAltContigFile` <file with names of contigs>}*

Specifying ALT liftover alignments to primary contigs

In a similar spirit to bwa-postalt.js, SNAP also supports lifting over alignments from ALT contigs to primary contigs as part of its paired-end aligner. We noticed cases when SNAP aligns reads well to an ALT contig (e.g., chr6) but poorly to another primary contig (chr8). In such cases, it may be better to use the ALT-to-primary liftover information (like DRAGEN which takes an additional --ht-alt-lifover sam file) to map the ALT alignment to its most likely primary contig location (chr6), instead of relying on SNAP's ALT-aware mapping. To enable SNAP's ALT liftover feature, an additional ALT-to-primary liftover file (e.g., hs38DH.fa.alt, part of bwakit) must be provided on the command line during indexing (`-altLiftoverFile`).

Currently, this feature works only in the SNAP alignment mode optimized for the DRAGEN variant caller (requires `-hc-` command line option to be specified during alignment)

Other indexing parameters

These parameters are unlikely to be useful, but this manual documents them for completeness.

You may change the number of threads used during bias computation and hash table build (the other phases are always single threaded) by specifying `-t`. Do not leave a space after the "`t`". If you specify more threads than the machine has cores, SNAP will only use one thread per core. By default, SNAP uses one thread per core.

SNAP uses closed hash tables, which means the keys and values are stored directly in the table rather than in a linked list of key-value pairs whose keys hash together. This kind of hash table is rarely used because it doesn't work well when elements are deleted. SNAP never needs to delete from the table, however, so it uses closed hashing to avoid an extra memory reference and cache miss on each lookup. To determine that a key isn't in a closed hash table it's necessary to scan the table from the first place the key might appear until an empty element is found. Therefore, it's necessary that there be some unused hash table entries. Having too few will make failed lookups take longer as they must scan farther. The *hash table slack* is the fraction of elements left empty. By default, it's 0.30, but you may change it using the -h flag. By reducing it you can make the index somewhat smaller at a performance cost. It's expressed as a floating-point number of empty slots per used one.

In SNAP's internal addressing of the reference it leaves some space between contigs and fills that space with Ns. This allows the critical path of the code to avoid making some bounds checks. You can change the default value of 500 bases with the -p. Specify the number of bases of padding directly after -p without leaving a space.

You can have SNAP put a histogram of seed popularity in a file by specifying -H. The filename comes directly after the H without a space. SNAP does not use this file, so only do this if you want to look at it yourself.

You can somewhat reduce the amount of memory used at index build time in exchange for a slower index build by specifying -sm. The max memory use in this case is about what SNAP will use when aligning with the index, so if `-sm` isn't enough you'll need a smaller reference, more memory, or smaller

key and/or location sizes (see above). Using this flag does not affect the content of the index, only its building.

`snap index` <input.fasta> <output directory> {`-t`NumThreads} {`-h` HashTableSlack} {`-p`Padding}

`snap index` <input.fasta> <output directory> {`-H`HistogramFilename} {`-sm`}

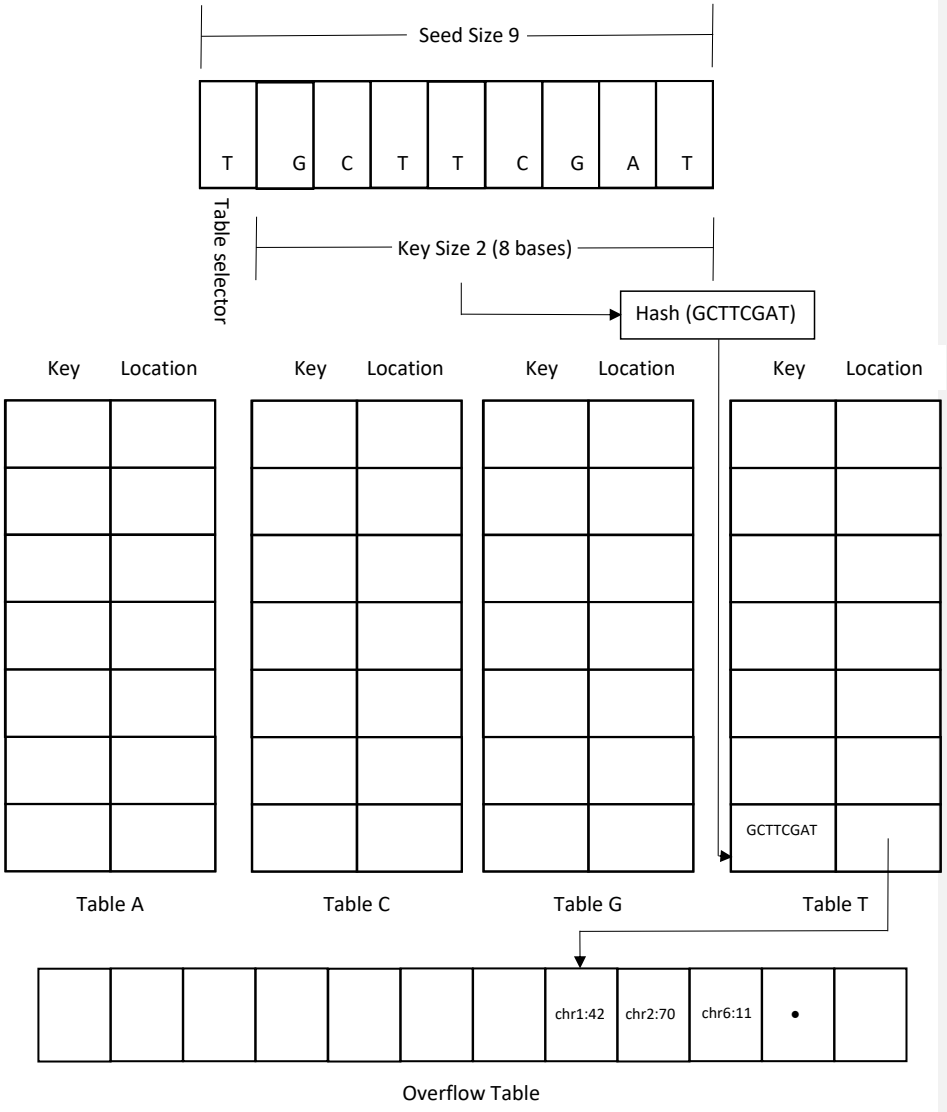

Figure 1: An example of the SNAP index with a seed size of 9 and a key size of 2. This results in the key being 8 bases, leaving one base to select which hash table, hence four hash tables. Lookup uses the table selector (T) to find the hash table, and then hashes the remainder of the seed (GCTTCGAT) to find the slot in the table. That slot has that key, so it proceeds to use the location. This seed occurs more than once, so it points to the overflow table, which maps it to chr1:42, chr2:70 and chr6:11.

Aligning

After generating an index, SNAP will align reads to the reference represented in the index. It has two modes: single- and paired-end. SNAP contains two different aligners for the two modes, though if the paired-end aligner is unable to align two reads as a pair (or thinks it did so poorly) it will fall back on the single-end aligner for those reads.

There is much that is in common between the two modes. This manual covers those things first.

The required part of the command line

Every alignment run of SNAP requires stating whether to run single or paired end and a path to an index, as described earlier, and one or more input files.

`snap single` <index directory> <input file> {<additional input file>}*

`snap paired` <index directory> <input file> {<additional input file>}*

For future examples that apply to both single and paired alignment (which is true of most of the options for SNAP) this manual uses single, but the options still apply to paired alignments.

While the examples all list the input files first on the command line, you may put them anywhere after the index directory, interleaving them with switches as you please.

File Types

SNAP can read and write many different file types. It can write to SAM and BAM. It can read from those, and from FASTQ, compressed FASTQ, interleaved FASTQ and compressed interleaved FASTQ (the latter two for paired-end alignments). Unlike other aligners, there is no need to convert formats before and after running SNAP.

In addition, SNAP can read from multiple input files which do not need to be of the same type. So, for example, SNAP can read input for a paired-end alignment from a pair of FASTQ files and also a BAM file without any preprocessing stages. Just specify the multiple inputs on the command line.

Unless instructed otherwise, SNAP will infer the file type from its extension. File names ending in .bam or .sam are SAM or BAM files, respectively. .fq and .fastq are FASTQ, and .fq.gz, .fq.gzip, .fastq.gz, and .fastq.gzip are GZIP compressed FASTQ files.

For file names that don't correspond to this convention, or for one of the rare file types that don't have a standard extension you may explicitly specify the file type. Do this by proceeding the filename with one of the following switches:

Switch	Meaning	Extension(s)
`-fastq`	FASTQ	.fq .fastq
`-compressedFastq`	GZIP compressed FASTQ	.fq.gz .fastq.gz .fq.gzip .fastq.gzip
`-sam`	SAM	.sam
`-samNoSQ`	SAM without the @SQ lines (nonstandard format)	none
`-bam`	BAM	.bam

-pairedInterleavedFastq	Paired, interleaved FASTQ (nonstandard format)	none
-pairedCompressedInterleavedFastq	Paired, compressed, interleaved FASTQ (nonstandard format)	none

SNAP can also read from stdin and write to stdout. To do this, use "–" as the filename. You must explicitly specify a type in this case, so "–bam –" would mean a BAM file on stdio.

If you want to have an input file whose name starts with a "-", you must explicitly specify the file type even if SNAP would otherwise infer it from the extension. This is because SNAP treats command arguments starting with a "-" to be switches unless it's otherwise obvious from context that they are not.

Interleaved FASTQ is a non-standard format for paired end reads. Rather than having separate files for each end they are one after another in a single FASTQ file. Compressed, interleaved FASTQ is the GZIP compressed version thereof.

SAM without @SQ lines is a nonstandard file format that is just what you'd think: SAM but with the header lines that specify the contigs omitted. It can only be used for output.

When using FASTQ for paired input the two paired ends must be consecutive input files, and if you specify an explicit file type, it must be before the first file and will apply to both of them.

Writing Output

Other than for performance measurements there's little reason not to write an output file. Nevertheless, SNAP's default is not to write output. To specify an output file, use the –o flag and a filename (with an optional file type specifier as described earlier). The filename may be "-" to write to stdout, in which case an output file type is mandatory.

snap single <index directory> <input file> –o {–outputFileType} <output filename>

Deciding how poor of an alignment to keep

The *edit distance* between two strings is the smallest number of single-character changes, insertions or deletions that are needed to change one into the other. For the edit distance computation, there's no penalty for opening an indel, and the per-base cost of indels is the same as for single character changes. This is not the metric that's used for selecting best alignments by default (see the affine gap section below), but it is used in determining how bad of a match to allow.

If SNAP can't find a sufficiently good alignment, it will mark the read unaligned. The –d flag specifies the largest edit distance to keep for matches. Increasing -d can have some performance reductions, but usually won't have much effect on reads that align well. In paired-end alignments the value of -d is the maximum of the sum of the edit distances for each read in the pair. The default value is 27; this is larger than the great bulk of indels present in human germline samples relative to the human genome.

snap single <index directory> <input file> {–d <max edit distance to consider>}

Looking for bigger indels

Ordinarily, SNAP will only look for alignments where the read and reference differ from each other by at most the -d value (see the previous section). Many of the cases where it makes sense to keep an alignment that's farther off happen when there's a big insertion or deletion in the read. While you could find these by increasing -d that would come at a considerable performance penalty. SNAP provides special support to look for longer indels. It does this by looking to see if the index lookups for the beginning and end of the read are slightly misaligned which would indicate an indel in the middle of the read. In those cases, it uses a more permissive max edit distance, which has a default value of 35.

Commented [BB2]: We used 40 for the runs. Should we make that the default?

`snap single` <index directory> <input file> {`-i` <max edit distance to consider for potential indels>}

Specifying how many seeds to use

SNAP selects some portion of reads as seeds to look up in the index. It tries to select seeds that overlap as little as possible. Looking at more seeds may help to align reads with lots of differences from the reference spread throughout the read. More seeds may also slow down progress through generating more work. In some cases SNAP will complete an alignment before using the maximum number of seeds if it can prove that that alignment (and its mapping quality) won't be changed by more work.

There are two ways to specify the maximum number of seeds to use. The first is to set it with `-n`. The second, mutually exclusive method is to set the "seed coverage" with `-sc`. Seed coverage is a floating point number that means the total size of the seeds used divided by the size of the read (or of the longer read for paired end). For example, with a seed coverage of 2, a seed size of 20 and a read size of 100 it would use at most 10 seeds, because 2 * 20 / 100 = 2. `-sc` can be useful with an input that has reads of varying length.

The default is a fixed maximum seed count (not coverage) of 25 for single end and 8 for paired end. Because of the way the paired-end aligner works it needs fewer seeds and slows down more with more seeds. Hence it has a smaller default.

`snap single` <index directory> <input file> {`-n` <max seeds> |[5] `-sc` <seed coverage(floating point)>}

Hits per seed

Many references are greatly repetitive. There is very little benefit in using seeds that occur lots of times in the reference, but there is quite a lot of work to use them. SNAP handles this by pretending it didn't look up seeds if it finds that the seed has too many hits (and looking up another seed to replace it). Changing the value of this parameter can have quite a large effect on performance, and it can also change alignment quality at the margins.

The `-x` flag tells SNAP's single-end aligner to look at all seeds, regardless of the number of hits, effectively setting `-h` infinite. This can result in quite poor performance unless coupled with `-f`, which tells SNAP to stop looking for better alignments as soon as it finds one that's good enough; this is useful for, for example, filtering out reads that match the human reference when trying to look at pathogens.

For paired-end alignments there are two values: one for the paired-end aligner and one for the single-end aligner that's used when the paired aligner fails to see a pair. Because the paired-end aligner only

[5] A vertical bar means to choose one or the other, but not both.

evaluates hits where there is evidence that both reads map near to one another it can often quickly dispense with hits. As a result, it can use a much larger limit without as much of a performance effect.

`snap single` <index directory> <input file> {`-h` <max hits to consider a seed>} {`-x`} {`-f`}

`snap paired` <index directory> <input file> {`-H` <max hits for paired-end>} {`-h` <max hits for when paired-end fails to find an alignment and it falls back to single-end>}

The default values are 300 for `-h` and 4000 for `-H`.

SNAP also has an option to ignore alignment candidates that don't appear in the hit sets for enough seeds. By default, this is one, *i.e.*, SNAP will consider alignments that are supported by only one seed. Increasing this value will improve alignment speed at the cost of some quality. Use the `-ms` switch to specify it.

`snap single` <index directory> <input file> {`-ms` <min seeds to support an alignment candidate>}

Threading

By default, SNAP runs one aligner thread per core of the machine that you're using. It also runs some threads to do input and output, though they often don't use much processor. Ordinarily, using this default is the right thing to do.

If, for instance, you intend to run other jobs simultaneously and wish to leave some cores for those tasks you might want to limit the number of threads SNAP uses. This might be particularly useful when SNAP is reading or writing from/to stdio and you need CPU for the other end of the pipe(s).

You specify the number of aligner threads with -t.

`snap single` <index directory> <input file> `-t` <number of aligner threads>

SNAP will bind threads to their cores. This prevents the operating system scheduler from moving them around, which would result in extra, unnecessary cache coherence traffic between cores and sockets would reduce performance. The `-b-` switch turns this behavior off. This effect is usually small.

`snap single` <index directory> <input file> `-b-`

SNAP will prefetch data from the index and reference genome. Depending on the other alignment parameters, this may improve performance by reducing the amount of time that a core stalls waiting for data to come back from memory. You can disable this by using the -P switch. Typically, this effect is small.

`snap single` <index directory> <input file> `-P`

Sorting, Indexing and Duplicate Marking

SNAP not only does read alignment; it also will sort, index, and mark duplicates. Doing this directly in the aligner saves the work of repeatedly reading and writing data to the file system. SNAP may also simply be faster than other tools at this job.

To sort, index, and mark duplicates, add the `-so` (sort) flag.

SNAP does not assume that the read set fits in memory, so it uses a merge sort. This works by sorting chunks of reads that fit in memory, writing out the sorted chunks and then merging them back together into a single sorted output file. The fewer chunks there are the faster the sort will run. By default, SNAP will use one gigabyte of memory per thread, which results in chunks of one third of a gigabyte[6]. You can change this by explicitly specifying the total (not per-thread) sort memory using the `-sm` switch. When you do this, it's necessary to make sure that the machine has enough memory both for the index (which can be quite large) and the sort buffer. If you use too much, either the machine will run out of memory and the alignment run will fail, or it will run out of physical memory, start paging, and have horrid performance.

We measured sort performance as a parameter of memory size, and for reasons we do not completely understand it seems to have a peak at 20GB (on our 16 core machine) and slow down thereafter. You should not necessarily assume that larger values of -sm will result in better performance.

By default, SNAP writes its sorted chunk files in the same directory as the final output file and deletes them when it is done. You may have it put them somewhere else (for either space or performance reasons) using the `-sid` (sort intermediate directory) switch.

SNAP sorts chunks while the alignment is running. It waits until an alignment thread has aligned a chunk's worth of reads and then sorts that chunk and writes it. If it wrote them out unsorted, it would have to read them, sort chunks, and write them before getting to the merge phase. It saves this extra work by doing the sort inline. Doing the sort this way reduces the speed of the alignment phase but speeds up the overall completion time.

By default, SNAP marks duplicates with sorting and (if writing BAM) indexes the resulting output file. You may disable one or both using the `-S` flag. Turn off indexing with an "`i`" and duplicate marking with a "`d`". Leave a space between the `-S` and the `i` and/or `d`. SNAP supports both paired-end and single-end read duplicate marking, is library-aware and matches the output from Picard MarkDuplicates, except on rare occasions when Picard MarkDup uses offset of records in the original file to resolve ties.

For each read aligned to reference location `loc`, all subsequent read alignments aligned to reference locations in the window `[loc, loc + MAX_READ_LENGTH + MAX_K]` are scanned to identify duplicates (to account for duplicate reads with soft-/hard-clipped alignments.).

`snap single` <index directory> <input file> `-so` {`-sm` <total sort memory in GB>} {`-S` {`i`} {`d`}} {`-sid` <sort intermediate directory>}

CIGAR strings

Part of SAM and BAM output is the CIGAR string, which describes the differences between the read and the reference against which it's aligned. Standard CIGAR strings only specify where there are insertions and deletions (indels) and leave single-base mismatches unmentioned. Regions of the read that have no indels are described using the "M" (alignment match) CIGAR op.

The SAM/BAM spec provides for an alternative to "M". This version uses "=" to mean an exact match and "X" to use a mismatched base without an indel. Many tools that work downstream of an aligner do

[6] Each thread gets three buffers. One is for reading into (which happens asynchronously with aligning). When the aligner fills a chunk, the filled chunk is sorted from one buffer into another, requiring two.

not recognize this format, so it is not SNAP's default. However, it is sometimes helpful when looking at alignments manually and some downstream tools may want it. You can switch to the alternate CIGAR format with the -= option.

`snap single` <index directory> <input file> `-=`

Minimum read lengths

SNAP will not try to align reads that are too small. Because it needs a perfect match between the read and the reference of at least one seed, too small reads may fail to align well. By default, reads below length of 50 are dropped. You may change this value with the -mrl (minimum read length) switch:

`snap single` <index directory> <input file> `-mrl` <minimum read length to align>

Index Loading

SNAP's index may be large and may take a long time to load, especially when SNAP is being used for metagenomics where the reference is much larger than the human genome. There are two ways to load it into memory: reading it from a file and memory mapping it. Reading is just what it sounds like: SNAP allocates memory for the index and reads the index files into that memory.

Memory mapping takes advantage of an operating system feature where it is possible to take a file and just make its contents be part of the address space of a process. This has some advantages. It means that the same physical memory is used for the operating system's file cache and for SNAP's memory. When SNAP exits, the index will still be left in the system cache, so if you run SNAP again quickly it will not need to be read from disk again. This is especially useful when memory isn't large enough to hold two copies of the index, one for the cache and one for SNAP. If, for some reason, you wanted to run multiple simultaneous copies of SNAP using the same index, they will share memory for the index if and only if you memory map it. By default, SNAP maps the index. Using the `-map` switch explicitly asks it to map; the switch mostly exists for compatibility with older versions that didn't map by default. `-map-` (with a trailing dash) says to read rather than mapping.

Because reading from disks (though not SSDs) is much faster when it's done sequentially, after SNAP memory maps the index it will read the index data sequentially to make sure that it's all in the system cache and not waiting to be read on-demand. On some versions of Linux (though not on Windows), reading in a file that's not already in the system cache by looking at memory mapped pages is much slower than reading it using IO. SNAP works around this problem with the `-pre` flag, which will read the file from the file system (and into the system cache) before mapping it. This is different from not memory mapping because there is still only one copy of the index (which is in the system cache). On Linux, `-pre` is the default, while on Windows it is not. To ask SNAP not to prefetch, use `-pre-` (with a trailing dash).

`snap single` <index directory> <input file> {`-map | -map-`} {`-pre | -pre-`}

Writing more than one alignment per read

SNAPs default is to produce a single best alignment for each read that its maps (if it can; otherwise, it leaves the read unaligned). It can, however, also write alternate alignments. Alternate alignments are written with the 0x100 (secondary alignment) flag set in the SAM/BAM output.

The `-om` (output multiple) flag tells SNAP to write alternate alignments for a read. `-om` is followed by a number that says how much worse (in edit distance) a secondary alignment can be and still be written. Specifying 0 for the value means to write all alignments that have the same number of edits as the best alignment and no others.

You can limit the number of alternate alignments written per input read with the `-omax` flag.

Sometimes (especially in metagenomics applications) it is useful to see if a read maps well to a contig, but the particular mapping is less interesting. You can limit the number of alignments to a given contig for a single read with the `-mpc` (max per contig) flag. `-mpc` filtering is applied before `-omax`. The `-mpc` flag limit counts the primary mapping as one mapping to its contig.

When computing which alternate alignments to keep, SNAP ordinarily applies a penalty for an alignment that lands off the end of the contig. The `-ae` switch turns this behavior off and has SNAP treat off-contig pieces of a read as soft clipped at scoring time.

`snap single` <index directory> <input file> `-om` <max edit worse than best> {`-omax` <maximum count of alternate mappings per read>} {`-mpc` <max mappings per contig per read>} {`-ae`}

ALT contigs

Some reference genomes have alternate versions for some sections, typically corresponding to common (-ish) haplotypes in the population. If SNAP were to treat these as just more of the ordinary genome, it might map reads with reduced mapping quality because it would find multiple alignments with about the same likelihood. Instead, SNAP treats these ALTs specially.

Which section of a reference are ALTs is determined at index-build time. The section on index building contains a subsection that describes how to control this.

When SNAP is mapping a read, it will prefer the non-ALT alignment unless the ALT alignment is better by some amount. How much better can be specified with the `-asg` (ALT score gap) flag and is measured in edits. The default value is 2.

When computing the mapping quality for an alignment to a non-ALT contig, SNAP ignores any hits to the ALT contig.

The `-ea` (emit ALTs) flag will cause SNAP to emit a mapping to an ALT contig if that mapping would have been the best mapping were it to a non-ALT contig. This mapping will have the 0x800 (supplementary mapping) SAM/BAM flag set and will have its mapping quality computed considering all potential mappings, both to ALT and non-ALT contigs.

You can disable ALT awareness with the `-A-` flag.

`snap single` <index directory> <input file> {`-asg` <number of edits better an ALT mapping must be to be the primary mapping>} {`-ea`} {`-A-`}

Affine gap scoring

SNAP uses an affine gap scoring policy for indels. Affine gap scoring is computationally intensive and is used selectively only for alignments that have been determined to be promising by the Landau-Vishkin edit distance aligner (i.e., alignments whose edit distance is less than or equal to the value specified as

part of -d). Most of the candidate reference locations for a read are filtered out by the Landau-Vishkin aligner as unpromising. As a result, the slowdown resulting from re-performing affine gap scoring for a few promising candidate reference locations is tolerable.

SNAP vectorizes affine gap scoring using Farrar's striped algorithm. Farrar's algorithm stripes adjacent query characters across different SIMD vectors to simplify data dependencies in the dynamic programming formulation. Farrar's algorithm computes the entire dynamic programming matrix and has complexity O(MN / W), where M – text length, N – pattern length and W – width of SIMD vector (e.g., 128-bit SSE can compute 8 cells in parallel, assuming 16-bit precision for scoring). However, computing the entire matrix is not necessary for most of the candidate reference locations, since typically after scoring the first few locations, based on Ukkonen's algorithm, SNAP reduces the maximum edit distance limit for scoring subsequent locations. To reduce computation with smaller edit distance limits, we extend Farrar's fully vectorized implementation to only compute cells that lie within a 2d + 1 band of the major diagonal of the dynamic programming matrix, where d – current maximum tolerable edit distance. By default, SNAP invokes the banded affine gap implementation when the length of the pattern to be aligned >= 3 * (2d + 1).

In each affine gap scoring round, we prefer to align the entire query string against the reference (global alignment). But in cases where the ends of the pattern poorly match with the candidate reference string, we may clip the ends of the query to prevent the variant caller from calling false positive SNPs. This leads to a semi-global alignment, i.e., gaps towards the end of the pattern are not penalized). Similar to BWA-MEM, clipping is applied only if the resulting score obtained is significantly greater than the score obtained using global alignment (i.e., by default, greater than clipping penalty = 5). Note that edits in the clipped region are still considered when scoring candidate reference locations and match probability is computed assuming the entire query string aligns at the candidate reference location. Clipped base information is used only when finally aligning the entire pattern against the best candidate reference location to prevent the variant caller from calling SNPs incorrectly in soft-clipped portions of reads that have been aligned correctly. If you are observing variant calls in soft-clipped portions of SNAP aligned reads, please explore if options in the variant caller to exclude soft-clipped bases from analysis (e.g., set --dont-use-soft-clipped-bases true for GATK Haplotype Caller).

SNAP enables affine gap scoring by default. You can disable affine gap scoring with -G- flag. If affine gap scoring is enabled, options are provided to customize scoring parameters. -gm is the score for match, -gs is the penalty for mismatch, while -go and -ge are the penalties for opening and extending a gap (I.e., insertion or deletion) respectively.

snap single <index directory> <input file> {-gm <score for matching base, default : 1>} {-gs < penalty for mismatch, default : 4} {-go <penalty for opening a gap, default : 6>} {-ge <penalty for extending a gap, default : 1} {-G-}

DRAGEN variant caller mode

The DRAGEN variant caller provides much better variant calling accuracy when compared to GATK's Haplotype Caller. Previous versions of SNAP left many reads unaligned when it found that it could not find a sufficiently good end-to-end alignment for the read. One of the main reasons for this is because SNAP's Landau-Vishkin based edit distance alignment does not support soft clipping of reads. We found

that forcing such reads to align with large soft clips (at the beginning or end of reads or both) significantly increased the number of false positive variant calls from GATK's Haplotype Caller. Furthermore, in many cases large indels were missed.

On the other hand, the DRAGEN variant caller can better handle poor alignments with large soft clips without increasing false positives. To increase the number of reads pairs mapped by SNAP, for each unmapped read pair from the paired-end aligner after the first edit-distance + affine gap scoring pass, we perform another alignment pass using a Hamming distance aligner that supports soft-clipping. Like in the first edit distance pass, Hamming distance-based scoring is used to find a few promising locations to align the read pair. These promising locations are further scored using a slow but accurate affine gap scoring method to determine the best location to align the read pair.

By default, SNAP outputs alignments that can provide the best accuracy with GATK's Haplotype Caller. For users that intend to take advantage of SNAP's alignment speed and pass SNAP's alignment output through DRAGEN's variant caller, we provide the `-hc-` command line option, which enables DRAGEN variant caller-specific optimizations in SNAP. Also included in the DRAGEN variant calling mode are a few read base-quality score aware optimizations that can improve the identification of small indels.

Mapping quality

SNAP estimates the likelihood that its mapping is correct and reports that value in the MAPQ field of the SAM/BAM output. It does this by finding mappings that are farther from the read than the best mapping, and estimating the probability that the read really came from one of the alternate mapping candidates.

How far beyond the best hit it looks is determined by the `-D` flag. The default value is 2. Because the probability of an incorrect alignment drops exponentially in the difference in alignment distance between the best and other alignments, there is little effect (other than reduced performance) of increasing the value of D. Conversely, reducing it will improve performance but reduce the accuracy of MAPQ.

`snap single` <index directory> <input file> {`-D` <edit distance beyond best hit to search>}

Output Filtering

SNAP has options that let you emit only some of the reads depending on their alignment results. You can filter to emit reads that are aligned, unaligned, aligned at MAPQ 10 or better, aligned at MAPQ less than 10, that are long enough to align (i.e., length greater than the minimum read length specified with -mrl), or too short to align. In addition, you can apply a filter to both ends of a paired read in paired-end mode.

There are two different ways to specify filtering. The common options are handled by the `-F` flag. `-F a` says to emit only aligned reads, `-F s` only reads aligned at MAPQ ≥ 10, `-F u` emits only unaligned reads and `-F l` emits only reads at or above the minimum read length. `-F b` says that the filter specified by `-F` must apply to both ends of paired-end reads for the read to be emitted. You can specify only one filter with `-F` (other than `-F b`).

`snap single` <index directory> <input file> {`-F a` | `s` | `u` | `l` {`-F b`}}

-F only lets you specify some of the possible range of SNAP's filtering behavior. The -E switch is fully general. `-E` says to emit only those reads that are of a type that it describes and to discard the rest. -E supports

- `s` - MAPQ ≥ 10
- `m` – MAPQ < 10
- `x` – too short to align
- `u` – unaligned
- `b` – emit reads only if both ends of a paired-end alignment meet the filter requirement

You may specify as many classes for `-E` as you'd like in a single `-E` switch.

Input Reads

SNAP may clip reads on input if the quality string for the read indicates to do so. SNAP clips all the bases from the end of a read where the base call quality is '#' if it's directed to do so. You can control this clipping behavior with the `-C`xy switch, where each of x and y are either a `+` or `-` to indicate whether clipping should happen at the front or back of the read respectively. Unlike most other SNAP parameters, there is no space between the C and the plus or minus.

The default is -C-+, that is to clip from the back only.

Parts of reads that are clipped will show up as soft clipped (CIGAR string S) in the output.

`snap single` <index directory> <input file> {`-C--` | `-C-+` | `-C+-` | `-C++`}

By default, SNAP ignores soft clipping of reads in SAM/BAM input files. Specifying `-pc` (preserve clipping) has it keep the input clipping.

`snap single` <index directory> <input file> {`-pc`}

When reading from a SAM or BAM file some reads may be marked as supplementary or secondary alignments. Among other things, these flags mean that there is a different copy of the same read somewhere in the input file that. By default, SNAP drops these reads on input. To keep them, specify `-sa`.

`snap single` <index directory> <input file> {`-sa`}

Read Groups

SNAP will preserve the read group specified in an input format (SAM and BAM) that specifies them assuming that they're present. For inputs without read groups (FASTQ) or for SAM and BAM lines that don't have read groups specified, SNAP allows setting the default read group for output. If you do not specify a read group, SNAP will use the "FASTQ" as the read group.

There are two ways of specifying read groups. The simple one is the `-rg` switch. With it you provide a string that will be the default read group and put on each read line with the RG tag.

The `-R` switch allows specifying exactly what to put in the output, including possibly extra optional fields. For this switch you can specify output tabs with `\t` (backslash t) and literal backslashes with `\\`

(double backslash). Other characters following the backslash are illegal. If you don't start this with "@RG\t", SNAP will add it. The string must also include an ID: field.

`snap single` <index directory> <input file> {`-rg` <read group> | `-R` <complete read group line>}

Performance Options

There are a number of switches that affect how SNAP runs but not the alignments that it makes. Most of these disable optimizations and are only intended to show the value of the optimization in question, but a number of others might be useful in some circumstances.

To run SNAP at low scheduling priority specify `-lp`. This instructs the operating system to allow other uses of the processor to run in preference to SNAP. Because SNAP often uses all of the available compute resources it may make the machine not as responsive as desirable, and this switch reduces the effect. This is only implemented on Windows.

`snap single` <index directory> <input file> {`-lp`}

SNAPs index is large and is not needed after alignment is complete and the reads are being sorted. SNAP will drop the index from memory for the sort if it gets the `-di` switch. However, memory demand is typically lower during sort than during alignment, so if you have enough memory to align you also have enough memory to sort.

`snap single` <index directory> <input file> {`-di`}

If you overuse the memory of a machine the operating system will start paging: it takes data that ordinarily would be in memory and writes it to storage. This is very, very slow and doesn't work well with SNAP, which has a largely random memory access pattern (since most of the memory is used by the index which is a hash table). SNAP has an option where it measures its own progress and if it thinks the system is paging it will kill itself. If this is happening, you should run on a machine with more memory, build a smaller index or use less memory for sorting. `-kts` ("Kill if too slow") enables this behavior.

`snap single` <index directory> <input file> {`-kts`}

The remaining performance options disable optimizations or alter timings in the alignment algorithms without changing the alignment behavior. Other than debugging or writing papers about SNAP there is little reason to use them. We include them only for completeness.

`-nu` disables the Ukkonen optimization and has SNAP put effort into computing edit distances that it knows will be too large to matter. `-no` disables the ordering in which possible mappings are evaluated. `-nt` disables an optimization where some candidate alignments can be eliminated solely based on the seeds that didn't hit at those locations. `-B` tells SNAP to wait until all threads have allocated memory before starting alignment. It's there to allow the developers to differentiate memory allocation time from alignment time. It is only available on Windows.

`snap single` <index directory> <input file> {`-nu`} {`-no`} {`-nt`} {`-B`}

Miscellaneous Options

When SNAP scores a read internally, it applies a penalty if the read would map off either end of a contig. However, when writing output this portion of the read will be soft clipped and the edit distance score

reported in the NM tag will not include the off-contig penalty. On request, SNAP will also include its internal score in a user-defined tag. Specify the `-is` flag followed by the two-letter output tag. As per the SAM/BAM spec, this output tag must start with `X`, `Y` or `Z`.

`snap single` <index directory> <input file> {`-is` <two letter output tag>}

SNAP can write an output file with some timing profiles. Tell it to do so and the output filename using the `-pro` flag.
`snap single` <index directory> <input file> {`-pro` <profile output filename>}

SNAP buffers its output for efficiency. If for some reason a single write (or, more likely, file header) won't fit into a single write buffer SNAP cannot continue. You can increase the write buffer size with the `-wbs` parameter. Its value is in megabytes and by default it is 16. There is no reason to change this unless you get an error message telling you to do so.

`snap single` <index directory> <input file> {`-wbs` <write buffer size in MB>}

Options that only apply to paired-end alignment

When reading an input of reads that have been aligned by a paired-end aligner (*i.e.*, in SAM or BAM format), SNAP must reconstitute the pairs so that SNAP's alignment can treat them as a pair. However, some input files have reads aligned without their mates being marked: they don't have the RNEXT and/or PNEXT fields filled in that allow SNAP to find their mates. Ordinarily, SNAP will simply discard these reads. However, SNAP can still find mates by looking at the read IDs. This may be slow and use a large amount of memory, but it is necessary when, for instance, you're trying to align as pairs a set of reads that had previously been aligned by a single-end aligner. The `-ku` (keep unpaired-looking) switch enables this feature.

`snap paired` <index directory> <input file> {`-ku`}

SNAP's paired-end aligner needs to know the possible range of inter-read spacing for pairs. It will look for alignments in this range, and if it's not able to find one (or not able to find one that looks good enough), it will fall back on aligning both reads with the single-end aligner and using the result. There are several flags that control this behavior. `-s` specifies the range of spacing between read ends. The default is 1 to 1000 bases. `-ins` tells SNAP to periodically infer the inter-read spacing based on the input. SNAP infers inter-read spacing using the algorithm from BWA-MEM. By default, after aligning a batch of a quarter million read pairs, SNAP begins to infer read spacing for the next batch. After filtering read pairs whose spacing bounds are too far away from the 25th and 75th percentile respectively, the mean (u) and standard deviation (s) of inter-read spacing is computed. The new spacing bounds for the next batch are set to be at least [u-4s, u+4s], but could be larger based on the distance between the 25th and 75th percentile spacing. `-fs` forces read pairs to be in the spacing range or leave them unaligned (this doesn't make much sense when combined with `-ins`, but it's still legal).

`snap paired` <index directory> <input file> {`-s` <min spacing> <max spacing>} {`-ins`} {`-fs`}

Because SNAP uses the single-end aligner for some reads when running a paired-end alignment, you may want to set the max seeds used for single-end separately than for paired-end. The `-N` option does this. The default is 25.

`snap paired` <index directory> <input file> {`-N` <max seeds for single-end aligner>}

After paired-end alignment, SNAP may reconsider performing single-end alignment for some read pairs, if:

(1) the read pair is aligned to an ALT contig, but the alignments are not significantly better than other non-ALT alignments for the same read pair. The `-es` option controls this, which is the minimum total edit distance by which a read pair aligned as ALT needs to be better than other non-ALT alignments for the same read pair. The default is 3.

(2) the read pair is aligned to a non-ALT contig and at least one of the reads in the pair is not too well aligned (i.e., has edit distance greater than a threshold). The `-en` option specifies this threshold. The default is 3.

SNAP only considers results from the single-end aligner if it has a significantly better affine gap score than the one returned by the paired-end aligner. The `-eg` option is used to determine the amount by which single-end alignments need to be better than paired-end alignments. The default is 15.

By default, SNAP does not enable Hamming-distance based soft-clipping optimizations for the single-end aligner. These optimizations can be enabled using the `-eh` option.

`snap paired` <index directory> <input file> {`-es` <min total edit distance of paired-end ALT alignment to be better than non-ALT alignments to skip single-end realignment>} {`-en` <max edit distance of reads from a paired-end non-ALT alignment to skip single-end realignment>} {`-eg` <min affine gap score by which single-end alignments need to better than paired-end alignments to be considered>}

`snap paired` <index directory> <input file> {`-eh`}

Running Multiple Alignments in a Single Execution of SNAP

There are two different ways to run more than one alignment in a single execution of SNAP: the comma syntax and daemon mode. This is useful to do because if consecutive runs use the same index SNAP will not need to reload the index for each alignment, which in some circumstances will save significant time.

The comma syntax

The comma syntax is useful when the set of alignments to be done is known ahead of time. It is a way to list different alignment jobs on the same command line and then execute them sequentially. To use it, run SNAP as you ordinarily would, and then at the end of the command line put a comma (separated from other parameters by spaces on either side), followed by another set of parameters (starting with "single" or "paired"). So, for example, to align reads1.fq and then separately reads2.fq you could do:

`snap single` <index directory> `reads1.fq -o reads1.bam , single` <index directory> `reads2.fq -o reads2.bam`

This will execute the jobs one after the other and will only load the index once, assuming that the same index is used for each job. You can specify as many jobs on a single command line as you'd like, subject only to the operating system's maximum command line length limit.

Daemon mode

Daemon mode is useful for circumstances where you do not know the jobs you'd like to run at the time you start SNAP, but would still like to avoid loading indices when possible. You run SNAP in a mode where it waits for commands, and then send it those command using the SnapCommand app. Like with the comma syntax, SNAP will not unload its index after completing an alignment, and if the next job uses the same index it will not reload it.

To start SNAP in daemon mode, do:

`snap daemon` {named pipe name}

The optional named pipe name parameter is the name of the communications channel that SNAP and SnapCommand will use. In Windows this is a named pipe and in Linux is it a Unix-domain socket. Typically you would not specify the pipe name and instead use the default. If you want to run more than one instance of SNAP in daemon mode on the same machine at the same time, at least one of them will need an explicit named pipe name (and they can't have the same name).

SNAP in daemon mode waits for commands. You can send it a command with the SnapCommand app. Run it on the same machine as SNAP and give it command-line arguments just as you would SNAP:

`SnapCommand` {`-p` named pipe name} `single` <index dir> `input.fq -o output.bam`

The `-p` option that specifies the named pipe name is optional, but must be supplied if you specified a named pipe name when you started SNAP in daemon mode.

In addition to the usual command, SnapCommand has the "exit" command, which will make the SNAP process exit:

`SnapCommand exit`

A given instance of SNAP will only process input from one SnapCommand at a time. If you run a second one before the first exits it will wait until the first alignment is completed and then immediately send the command.

When SNAP is running in daemon mode, SNAP's output is written by both SNAP itself and the SnapCommand app.

Input and output to stdio is not supported in daemon mode.

File path names sent in SnapCommand are processed in the working directory of the SNAP program, not SnapCommand. SnapCommand just passes the parameters as text to SNAP, it does not interpret them.

Index