User's Guide to SeqNinja™ (Command-Line Edition)

DNASTAR, Inc. 2014

Contents

Before You Begin

We're here to help! If you have any difficulties with or questions about this application, please contact a DNASTAR support representative:

- E-mail: support@dnastar.com
- Phone (Madison, WI, USA): 608-258-7420
- In the USA and Canada, call toll free: 1-866-511-5090
- In the UK, call free on: 0-808-234-1643
- In Germany, call free on: 0-800-182-4747

This user's guide pertains to the command-line version of SeqNinja™, a component of DNASTAR's Lasergene® Core Suite version 12. This document was last updated on September 29, 2014.

If you accessed this help from within a Lasergene application: The help PDF installed with your application was current at the time of the version release. Click the following links to view the most current PDF or online NetHelp for the SeqNinja Command-Line application.

To access free video tutorials for DNASTAR products, please click on www.dnastar.com and use the tabs at the top to choose **Support > Videos**.
For copyright and trademark information, please see the Legal Information page of our website.

Overview

SeqNinja supports automation of repetitive tasks via a custom scripting language that manipulates sets of sequences. This can be useful both for reducing effort, and for repeatability of an operation on updated data. For example, SeqNinja allows you to:

- Read data from and write data to a variety of supported file formats.
- Convert from one file format to another (e.g., GenBank to FASTA).
- Generate subsequences with annotations in the subsequence's coordinate space.
- Locate all genes annotated with a specified value or property.
- Extract a set of annotated features from a genome and save them as protein or nucleotide sequences.
- Create a circular permutation of a linear presentation of a circular genome.
- Reverse complement a sequence or set of sequences.
- Translate a sequence or set of sequences.
- Break up a genome into pieces of a specified length with a specified overlap, useful for simulations.
- Migrate annotations between different versions of a genome and convert genome coordinates between assemblies.
- Merge contigs into a pseudo-molecule with a linker, and later break that pseudo-molecule back into individual contigs. This ability to concatenate and de-concatenate is useful for gene-finding.

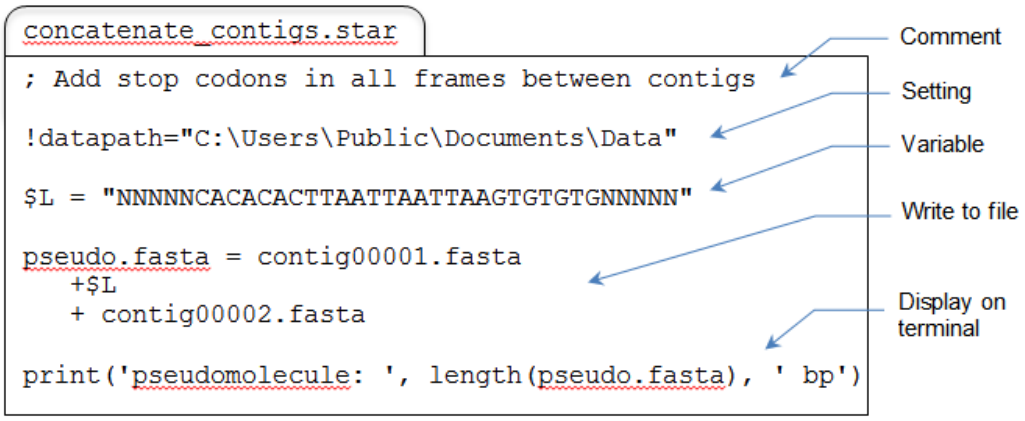
To learn how to use SeqNinja, see the next topic, *Getting Started with SeqNinja*.

Note: For more on the origins of SeqNinja, see *Research References*.

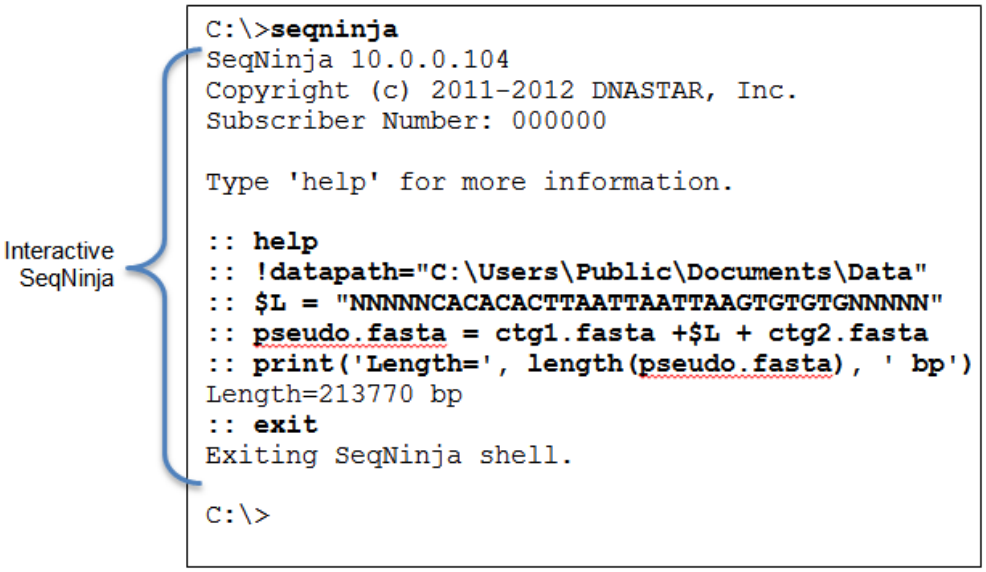
Getting Started with SeqNinja

The command-line version of SeqNinja runs within a command shell (terminal) window. SeqNinja is launched in different ways, depending whether you'd like to run in "batch mode" or "interactive mode." All scripting language commands can be used in either mode.

- In batch mode, you run an "argument" (command script) stored in a file. Batch mode is useful for complex scripts and for repeatability. The script file provides a record of the operations performed. An example of a script for batch mode use is shown below:



- In interactive mode, you enter and run commands one-at-a-time, directly in the SeqNinja command shell. Interactive mode is useful for experimentation and brief, *ad hoc* commands. A sample interactive section is illustrated below.



Running SeqNinja in batch mode:

From within a Command Prompt window (Win) or Terminal window (Mac), type "`seqninja`" followed by the name of a script, or by another command-line option.

If running a script, you will first need to prepare a file of scripting commands using any text editor and save the file with the extension .star (e.g., "*myfile.star*"). In the command shell, enter the command: `seqninja filename.star` and press **Enter**. This will run all the scripting commands in order of their appearance in the text file.

Running SeqNinja in interactive mode:

Launch the SeqNinja command shell in either of these ways:

- From within a Command Prompt window (Win) or Terminal window (Mac), type "`seqninja`."
- (Win only) From within a file explorer, double-click on the SeqNinja executable (located in the SeqNinjaCL subdirectory).

Once the SeqNinja shell has been launched, you will see a double-colon (i.e. "::") command prompt. You are now ready to begin entering scripting commands and pressing **Enter** to perform each operation.

Note 1: If you launch the SeqNinja command shell from outside a command shell (terminal) window, a command shell window will appear automatically. When you quit that SeqNinja session, the command shell window will disappear.

Note 2: The SeqNinja command shell does not support history or editing, other than backspace and "paste."

Notification of scripting errors:

If there are issues with a script, SeqNinja will attempt to flag these prior to executing the script. SeqNinja checks the script for issues such as:

- Input files that do not exist, are not readable by the user, or are not created by the script before they're used as input.
- Output files that are not writable by the user.
- Undefined variables.
- Unknown functions.
- Missing or unexpected function arguments.

Editing in the SeqNinja Shell

When using the SeqNinja shell in Macintosh, editing shortcuts such as **Cmd+A** (Select All) and **Cmd+V** (Paste) work as expected. When using the SeqNinja shell in Windows, however, these shortcuts (**Ctrl+A**, **Ctrl+V**, etc.) are disabled. To access editing commands from the shell in Windows, right-click on the header bar at the top of the Command-Line window and choose **Edit > [command-name]**.

- **Edit > Select All** – Selects all text in the current Command-Line session.
- **Edit > Paste** – Pastes the contents of the clipboard in the current line.
- **Edit > Find** – Launches the Find dialog.

Command-Line Options

The options below function from within the command shell, rather than from within the SeqNinja shell.

Objective	Expression	Comments
To enter the SeqNinja shell or to execute a batch mode command	`seqninja`	When used alone, launches the SeqNinja shell, allowing you to begin an interactive session. When followed by the name of a file or with one of the commands below, you will stay in the command shell and SeqNinja will execute the command or script in batch mode.
To display a usage description	`-h` `-help`	Displays a short usage description. Using this command does not cause you to enter the SeqNinja command shell.
To check the version number	`-v` `-version`	Displays the installed version number. Using this command does not cause you to enter the SeqNinja command shell.

Note: Using these expressions from within the SeqNinja shell may generate an error message.

Supported File Formats

SeqNinja can both read and write to any of the following file formats:

File type	Extensions recognized	Comments
FASTA single and multi-sequence files	.fasta, .fas, .fna, .fap	For more information, see NCBI's Accepted Input Formats page.
GenBank single and multi-sequence files.	.genbank, .gb, .gbk, .genpept, .gp	In a GenBank multi-sequence file, each sequence is terminated by a double-slash sign (//). For more information about GenBank flat file format, see NCBI's GenBank Flat File page.
SeqNinja set files	.star	SeqNinja can read, write, and execute set files, which are themselves editable. Sequences can be removed or re-ordered, though re-ordering sequences in large datasets can significantly slow processing time. A union of sets can be created by concatenating two set files.
DNASTAR and Lasergene sequence files	.seq, .pro, .mseq	Lasergene DNA and protein files have the extensions *.seq and *.pro, respectively. DNASTAR multiple-sequence DNA files have the extension *.mseq, and can be created by EditSeq, SeqMan Pro, and MegAlign.

The SeqNinja Language

Language Overview

The SeqNinja language supports comments and statements. Statements include *actions*, *assignments* and *settings*.

Actions are viewed as commands.

Assignments store data in a file or variable.

Settings configure options used implicitly throughout the rest of the script.

The following list outlines general conventions of the SeqNinja scripting language:

- In a script file, a statement may be spread over multiple lines, but may not be placed on the same line as another statement.
- Variables (e.g., file paths, etc.) are only stored in SeqNinja's memory until the end of the session, or until overwritten, whichever occurs first.
- White space and comments are always ignored.
- Single-line comments can be added after typing the characters ; or //.
- Multi-line comments can be added after typing the characters /*or*/.
- SeqNinja supports non-ASCII characters in filenames, but non-ASCII characters may not be used as variables.
- Scripts with non-ASCII characters, including international characters, must be saved with the UTF-8 encoding, without byte order marks.
- Double-quotes should be used around individual file paths, whether in settings or sequence expressions. Double-quotes *inside* of paths are never supported.
- Single-quotes should be used to signify a file pattern.

Escape Codes

All quoted strings (e.g., those that can be used with `print`, for example) accept certain "escape codes." These escape codes indicate that SeqNinja should put a tab, a new line, a quotation mark, etc. in the specified location:

Escape Code	Effect
^t	Inserts a tab.
^n	Inserts a line break.
^^	Inserts a caret character (^).
^'	Inserts a single quote into a string that is bracketed by single quotes.
^"	Inserts a double quote into a string that is bracketed by double quotes.

Example input:

Using `print` with the following argument causes the name of the file, length of the sequence, and the sequence itself to print in the output stream in a neatly organized manner.

```
print( 'sequence file name:^t', 'contig01.fas^n', 'sequence
length:^t', length("contig01.fas"), '^n', 'sequence', '^n',
"contig01.fas" )
```

Example output:

```
sequence file name:     contig01.fas
sequence length:        116581
sequence
TATGTCAGTAATTACCGCGTTCGCCAGCGTCAGTTCTCTGGCATTTTTGTCGCGCTGGGC
TTTGTAGGTAATGGCGTTATCACGGTAATGATTAACAGCCCATGACAGGCAGACGATGAT
GCAGATAACCAGAGCGGAGATAATCGCGGTTACTCTGTTCATTGCTGACCCCACAAACAG
ATTTCACGGTCAATCTCACGACGAGTCATGAGACCTTTCCATTGCTTACCGCCAGCATAT
GTCCAGCGACGTAGCTGATCACATGCGCCTTTGATATCGCCCTGGTTTATTTTGCGAAGA
```

File Patterns

A file pattern is a search string with one or more wildcards in the filename. The collect sequence function and split action both support file patterns as arguments.

The following are some rules pertaining to the use of file patterns:

- A question mark (?) wildcard matches exactly one arbitrary character. An asterisk (*) wildcard matches zero or more arbitrary characters. Split only allows asterisk (*) wildcards. Collect allows both asterisk (*) and question mark (?) wildcards.
- A file pattern may be relative or absolute. Relative paths in patterns are resolved in the same manner as for double-quoted path names.
- Wildcards can only be used in the filename, and nowhere else in the path.
- Single-quotes must be used to distinguish file patterns from individual file paths, which are double-quoted. This makes it possible to reference files whose names contain wildcard characters.

Settings

Settings configure values used implicitly throughout the rest of the script, and always begin with an exclamation mark (!). The value of settings may be a single word, or a quoted string using single or double quotes. In a setting, double-quoted strings are not interpreted as sequences. Settings include the following:

Objective	Expression	Examples	Comments

To set the root directory for relative file paths used in the rest of the script	`!datapath`
	
	`!datapath:in`
	
	`!datapath:out`
		`!datapath= 'C:\Users\MyName\ data\'`	Relative path names are presumed relative to one of the following, in order:
			
			• If set, `!datapath:in` (for input files) or `!datapath:out` (for output files).
			• Otherwise, if set, `!datapath`.
			• Otherwise, the working directory.
			
			Using different directories for input and output reduces risk of accidentally overwriting an input file.

To specify a temporary files directory	`!tempdir`	`!tempdir='D:/temp'`	This can be used to put temporary files on a drive with more space, or in a more accessible location.
To filter output features to those listed	`!features`	`!features='CDS, source'`	See the notes below this table.
To set verbose output of the types specified in the value	`!verbose`	`!verbose='TRACE, DATA'`	This sets verbose output of the types specified in the value. This can be useful for providing progress information and for trouble-shooting.

Objective	Expression	Examples	Comments

To output an auxiliary file to preserve features during format conversion	`!conversion`	`!conversion=natural`
		
		`!conversion=full`
			See the next topic, *Converting Between Annotated and Featureless Formats*. Also see the notes below this table.

To specify whether to record a sequence history in the comments section of generated sequences	`!comments:history`	`!comments:history='on'`
		
		`!comments:history='off'`
			Sequence history is only written when comments can be written to the output format. A sequence history can be written for GenBank files, Lasergene files, and FASTA files (when !conversion='full' is set).
			
			A sequence history includes:
			
			• When the sequence was created.
			• The name of the sequence, if it has a name before written.
			• A SeqNinja expression that can be used to create the sequence.

Note 1: Other settings can affect the display of features resulting from use of `!conversion` and `!features`. For example, including `!features=CDS` above a statement with range endpoints will cause only CDS features to be displayed.

Note 2: If you wish to set endpoints based on features that you want to later filter from the output, temporarily set `!features=*`. After setting the endpoints, change the `!features` setting to the type required for the desired output.

Converting Between Annotated and Featureless Formats

The default "!conversion" setting of *natural* causes SeqNinja to write/read only those items that are "natural" to the receiving/source file type. Changing the setting to *full* when changing from an annotated format to a featureless format and back again creates auxiliary file(s) to preserve features and comments that would otherwise be lost.

Note: The conversion setting must appear in the program *before* the operation in which file output is specified.

---

1) Converting from GenBank and Lasergene (*.seq) format to FASTA format

"*Full*" can preserve features and comments that would normally be lost when converting from annotated format to FASTA format, then back again. However, "*full*" is not suggested in all cases; for instance, it is not recommended when you plan to make sequence modifications (other than substitutions) while the sequence is in FASTA format.

When "*full*" is selected, SeqNinja outputs three files during conversion from an annotated format to FASTA format:

- *.fasta – This is an standard FASTA file containing sequence reads. When writing FASTA files from a source that contains comments, the FASTA header line includes the first line of comments from the source. For example:

```
 > NC_010473 LOCUS  NC_010473 4686137 bp  DNA circular BCT
04-OCT-2012
```

- *.fasta.features –This file contains tab-delimited feature information from the original annotated file, and can be opened and modified using a spreadsheet utility such as Microsoft Excel®. The modified file can be read by SeqNinja, with the following restrictions: 1) Sequences must appear in exactly the same order as in the corresponding FASTA file. 2) The first six columns must remain present in the original order. Subsequent columns contain qualifier values named in the column header, and can be removed or amended, as desired. 3) The modified file must be saved as tab-delimited text, not as an.xls or .xlsx file, and its name must remain unchanged.

- *.fasta.comments – This file contains one comment pertaining to each sequence in the FASTA file, and in the same order.

Note: Comments are maintained during file conversions (e.g., *a.seq* = *b.gbk*). However, comments are discarded during sequence transformations (e.g., *a.gbk* = *b.gbk* + *c.gbk*).

---

2) Converting the saved FASTA file back to GenBank or Lasergene format

If you used *full* during the conversion to FASTA, you can convert back to GenBank or Lasergene, using *full* again, without losing the original features and comments. As long as you keep the auxiliary files in the same folder as the FASTA data file, SeqNinja will automatically place the features and comments back in the GenBank or Lasergene file.

Integral Expressions

Supported integral operations are shown below, and always begin with a percent sign (%).

Objective	Operators	Examples	Comment
To add, multiply and divide	`+,*, /`	`%i=1+2*4`	Operands and results are always integers, and any fraction obtained through division is discarded.
To make an assignment	`=`	`%i=1+2`	This overwrites any earlier assignment.
To assign a variable to an integral expression	`%variable_name`	`%i=3+4+%j`	Legal characters in a variable name are A-Z, a-z, 0-9, and _ (underscore).

To measure length	`length`	`%len=length("H:\MG1655.fasta")`
		
		`print(length($M+$A))`
			If the length argument contains multiple sequences, only the first is used.

To count sequences	`countSequences`	`%s=countSequences ("H:\454.fasta")`	Total count across the entire set of sequences provided.

Sequence Expressions and Assignments

A sequence expression describes a sequence or a set of sequences. Sequence expressions always begin with a dollar sign ($) and include:

- A sequence filename, a literal sequence, or a variable representing the value of a sequence expression.
- A reference to a single sequence in a multi-sequence file by path/name, name or number.
- A fragment (subrange) of a sequence, where each coordinate is specified by a "site expression" (also see *Integral Expressions*).
- A concatenation of sequence expressions.
- A function of a sequence expression, such as *complement*, *cut*, *extract* or *translate*.

Sequence Filenames, Literal Sequences and Variables

One or more sequences can be obtained from a filename or variable, or provided as a literal sequence. Sequence variables always start with a dollar sign ($).

Objective	Expression	Examples	Comments
To write any sequence expression to a sequence file	`filename=sequence_expression`	`myfile.fasta=$b`	Causes output to be written to the specified file, in a format determined by the specified file's extension. If you use the = expression and the filename already exists, you will be asked whether you wish to overwrite it. Appending is supported via the += expression (see next row).
To append data to an existing sequence file	`filename+=sequence_expression`	`myfile.fasta+=$b`	This adds data to an existing file rather than overwriting the existing data.
To convert to another format	`sequence_setA=sequence_setB`	`alpha.fasta=alpha.gb`	
To convert a set into a data file	`sequence_setA=sequence_setB.star`	`my.fasta=my.star`	

Objective	Expression	Examples	Comments
To assign a sequence expression to a variable	Simple filenames	`$a=myfile.fasta`	Double-quotes are necessary for filenames that include characters other than A-Z, a-z, 0-9, the underscore, or '.' after the first character.
	Filename with spaces	`$b="my filename.fasta"`	
	Filename with slashes	`$c="C:\data\MG1655-e-coli-k-12substrands.fasta"`	
	Unicode filename	`$d="Мое имя файла.fasta"`	
	Filename with diacritic mark	`$e="ma séquence.fasta"`	
To assign a variable or filename to a literal sequence	`"sequence"`	`$f="ACGT"`	Where a sequence expression might occur, double-quoted string literals are interpreted as either sequence literals or sequence filenames, depending on whether they contain characters outside A-Z and a-z.
	`sequence_set="sequence"`	`alpha.fasta="ACGT"`	

Note 1: The output file format depends on the filename extension on the left-hand-side of the assignment. A recognized filename extension is required.

Note 2: Within the same program, you may use the output sequence from an earlier step as the input sequence for a later step. However, the result might not be what you expected or desired. For example, if an extracted CDS initially had a /codon_start annotation (i.e., it did not start with an intact codon), that information would be lost in the extracted FASTA file. Then, when the file was subsequently translated, it would be in the wrong reading frame.

Multi-Sequence Files and Expressions

SeqNinja can read and write files that contain multiple sequences. For example, a sequence expression might be the name of a file containing multiple sequences. Some operations support multiple-sequences, while others operate only on the first. For example, reverse-complement supports multiple sequences. But concatenation works only on the first sequences of its operands.
To specify a single sequence from a multiple-sequence file, use one of these forms:

Objective	Expression	Examples	Comments
To pick a sequence by numerical index	`filename#index`	`"WIS_YPE_1.fasta"#2` `"WIS_YPE_1.fasta"#2(1,20)`	The first sequence in the file is index 1.
To pick a sequence by name	`filename#name` `filename#"name"`	`"WIS_YPE_1.fasta"#"Chromosome_contig2"` `"WIS_YPE_1.fasta"#"Chromosome_contig2"(1,20)`	This form works best if all the sequences in the set have unique names. If multiple sequences have the same name, then this expression will match the first encountered sequence whose name matches.
	`filename/sequenceName`	`"WIS_YPE_1.fasta"/"Chromosome_contig2"`	This refers to the sequence in the file as if it were a file in a directory.

Note 1: Some special characters in sequence names may require use of the # notation.

Note 2: Accessing multiple individual sequences by index from a multi-sequence file can be faster when the access is done in the order those sequences appear in the file. For example, if a file contains sequences (a,b,c,d,e,f), then accessing (a,c,f) may be faster than (f,c,a).

Sequence Fragments

Following some sequence expressions, a fragment (sub-range) can be specified using coordinates. Here are some rules pertaining to sequence fragments:

- Each coordinate is specified by a "site expression."
- Integral coordinates are supported. The first base has index 1.
- The full expression is always evaluated left-to-right.
- A sub-range applies to all sequences in a multi-file sequence. Ranges are re-evaluated for each member of the set to which they are applied.
- IUPAC ambiguity codes are recognized both in the sequence and in the expression. For example, typing AAS (where S = C or G) into an expression will cause SeqNinja to look for the first instance of AAC or AAG in the sequence. Conversely, typing AAC into an expression will cause SeqNinja to look for the first instance of AAC, as well as any combination of bases and ambiguity codes that would allow for AAC (e.g., AAS, WWM, etc.).

Objective	Expression	Examples

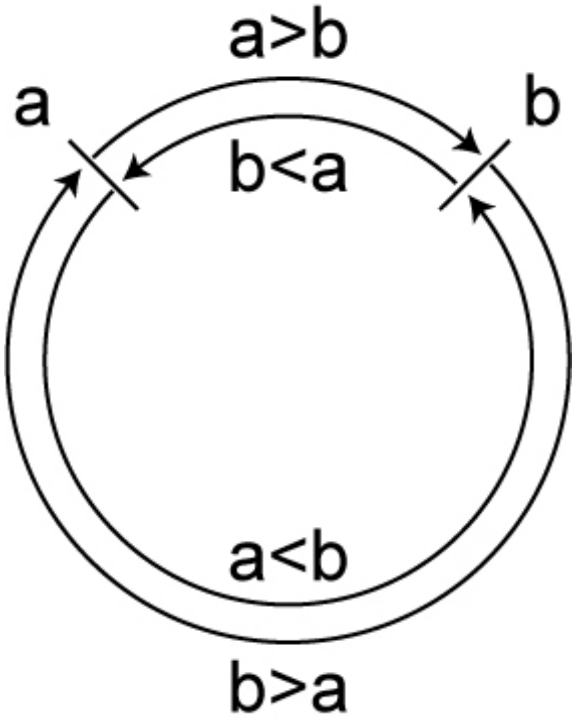
To read right from coordinates a to b, inclusive	`(a > b)`
	
	`(a,b)`
		`$B=alpha.fasta(1,414)`
		`$F=alpha.fasta(%i+3, 414)`
		`$C1=$A(1,414)=$A(1 > 414)`

To circle right	`(b > a)`	`$E1=$A(414 > 1)`
To circle the reverse complement	`(a<b)`	`$E2=$A(1<414)`
To read the reverse complement left from coordinates b to a	`(b<a)`	`$D1=$A(414<1)`

To specify the left end and right end bases	`lend`
	
	`rend`
		`$D=$A(lend,rend)`
		
		`$D=$A(lend+1, rend+1)`

To reference a base position relative to the left end of the sequence. (Note that "left" can only be used to specify the coordinate on the right-hand side).	`left`	`out.fasta = in.fasta("ATG", left + 12)`

See the image below for a visual representation of the different sequence segments.



Coordinates can also be specified by the results of one or more searches for matching sequence. The value of a search is a coordinate. Site expressions can include both searches and integral expressions, all evaluated left-to-right. In these examples, $A="AAAGGGCATCCCTTT":

Objective	Expression	Examples	Result
For left position, start at lend, search forward	`(lend > > "sequence", rend)`	`$A(lend > > "CAT",rend)`	CATCCCTTT
For right position, start at lend, search forward	`(lend,lend > > "sequence")`	`$A(lend, lend > > "CAT")`	AAAGGGCAT
For left position, start at rend, search reverse complement	`(rend<<"sequence", rend)`	`$A(rend<<"ATG", rend)`	CATCCCTTT
For right position, start at rend, search reverse complement	`(lend,rend<<"sequence")`	`$A(lend, rend<<"ATG")`	AAAGGGCAT
Search forward implicitly from lend	`("sequence", rend)`	`$A("CAT", rend)`	CATCCCTTT
Search forward implicitly from left position in direction/strand of single arrow	`(lend+number > "sequence")`	`$A(lend+3 > "CAT")`	GGGCAT
Search forward implicitly from right position in direction/strand of single arrow	`(rend<"sequence")`	`$A(rend<"ATG")`	AAAGGGATG
Look forward for first "CC", then look forward from there to next "T" (chained search)	`(lend > > "seq a" > > "seq b", rend)`	`$A(lend > > "CC" > > "T", rend)`	TTT
Find ambiguity code matches	`("sequence", rend)`	`$A("CAN", rend)` `$A("CAW", rend)`	CATCCCTTT
Search for "CAT" and add 3; this mixes search operators with arithmetic	`("sequence"+number, rend)`	`$A("CAT"+3, rend)`	CCCTTT

Sequence Concatenation

Sequence expressions can be concatenated with the '+' operator. Concatenation links together corresponding pairs in the left and right sequence sets. Following are some rules concerning sequence concatenation:

- When one of the sets is a singleton (for example, a literal), concatenation results in prepending or appending that single sequence to every sequence in the other set.

- If a concatenation includes more than one fragment from the same source sequence, then feature reassembly occurs when writing to a file format supporting features. Feature reassembly takes parts of the same feature in multiple fragments and combines them back into a single feature. When features are cut, reassembled, or (in some cases) offset, SeqNinja flags this with an annotation of the form: */note="***needs review***Explanation*, where *Explanation* is a short description of a potential issue (e.g., that the head or tail of a feature has been cut by a number of residues).

- A sub-sequence can cut a feature. Multiple sub-sequences concatenated together can temporarily result in multiple fragments of the same feature. SeqNinja can reassemble these fragments so that the output sequence has a single feature, and such fragments are reassembled if they do not overlap. Reassembly is performed left-to-right in the resulting sequence. For example, the "source" annotation in GenBank files can be cut into fragments by a SeqNinja expression. Before writing an output file, SeqNinja reassembles these fragments, as long as they don't overlap. If a concatenation does not include multiple fragments from the same source, feature reassembly does not occur.

Expression	Example	Comment
+	`beta.fasta="ATG"+alpha.fasta+"CTA"`	Adds a prefix and suffix to another sequence.
	`gamma.fasta=alpha.fasta(31,41)+(59,265)+(358,979)`	Concatenates fragments from the same source without repeating the source name.

Sequence Functions

Functions resulting in sequence expressions always start with a dollar sign ($) and include:

Objective	Expression	Example	Comments

To reverse-complement sequences	`~(sequence_set)`
	
	`Complement(sequence_set)`
		`$rc=~(foo.fasta)`
		
		`foo_rc.fasta=complement(foo.fasta)`
			Though the command is "complement" (for brevity), SeqNinja is actually calculating the *reverse complement* of the selection.

To assign a file to the reverse-complement of another file		`MG1655_rc.fasta=~("C:\data\MG1655_e_coli_k_12substrands.fasta")`	
To cut a sequence into smaller pieces	`cut(sequence_set, size)`	`bar.fasta=cut(foo.gb, %i)`	
To specify an overlap when cutting a sequence	`cut(sequence_set, size, offset)`	`bar.fasta=cut(foo.gb, 180, 60)`	This can be used to create faux reads from an assembled sequence.

To extract sub-sequences from sequences corresponding to the given features	`extract(sequence_set, 'feature_type[,...]')`	`bar.fasta=extract(foo.gb, 'CDS')`
		
		`bar.fasta=extract(foo.gb, 'CDS,gene')`
			Single quotes are required for arguments other than the sequence set.

Objective	Expression	Example	Comments

To extract matching features	`extract(sequence_set, 'feature_type:/tag="value"[,...]')`	`bar.fasta=extract(foo.gb, 'CDS:/gene="thrC"')`
		
		`bar.fasta=extract(foo.gb, 'CDS:/gene="thrC",CDS:/gene="thrL"')`
			When extracting matching features, a qualifier can be optionally specified. If no qualifier is specified, the result includes all features of the given type. If a qualifier is specified, the result includes features that include a matching qualifier.
			
			Note: Writing many small extractions to a file format supporting features can be slow. For each extraction, all of the features are evaluated for intersection.

To extract matching features using wildcards		`bar.fasta=extract(foo.gb, 'CDS:/gene="thr?"')`	Wildcards can be used in the qualifier value. A '?' matches exactly one arbitrary character, and a '*' matches zero or more arbitrary characters. The example matches CDS features with four-character gene names beginning with "thr".
To ignore source sequence qualifiers	`translate(sequence_set , '/codon_start=ignore')`	`$A=translate("myfile.gb", '/codon_start=ignore')`	Including "/codon_start=ignore" in the qualifiers causes any "/codon_start" qualifiers in the source sequence to be ignored.

Objective	Expression	Example	Comments

To translate all source sequences from DNA/RNA to protein	`translate(sequence_set)`	`$A=translate("myfile.fasta")`
		
		`$B=translate("myfile.fasta", '/transl_table=11')`
		
		`Test.gb=translate("myfile.gb", '/transl_table=11')`
		
		`$C=translate( "myfile.fasta", '/transl_table=VERTM' )`
			If the input sequence is annotated, each CDS feature will be translated separately, and any translation table and/or codon_start annotations will be honored. If the input is unannotated, or contains no CDS features, the entire sequence will be translated. Note that translation only progresses with sequences with lengths that are multiples of three (i.e., codons); an "extra" base or two at the end will not be reflected in the output.
			
			The standard code is used as a default unless specified differently in the file or in the qualifier overrides. The first codon in a sequence is translated as a start codon, if recognized as such in the genetic code. Otherwise, the default translation is used.
			
			Translation of a DNA or RNA sequence with ambiguities might result in an amino sequence with ambiguities, or not. For example, the result of translate ("RAT") is the ambiguity "B", but the result of translate ("ACN") is "T".

To override defaults or values specified in the file	`translate(sequence_set, '/tag=value [,/tag=value...]')`	`$A=translate("myfile.gb", '/transl_table=11,/codon_start=ignore')`	This argument is a comma-separated list of qualifiers.

Objective	Expression	Example	Comments

To mark the sequences in a set as being DNA, RNA or protein.	`dna(sequence-set)`
	`rna(sequence-set)`
	
	`protein(sequence-set)`
		`example.gb=protein("myfile.fasta")`
		
		`bar.fasta=protein("myfile.fasta")("NAN",rend)`
			This ability may be useful for sequences originating in formats where type is unspecified, such as FASTA files.
			
			If you use any of these functions, the information will be added to the output file, where allowed (e.g., *.gbk). The presence of sequence type information may affect the results of searching in an endpoint expression. For example, in DNA, "N"=anything, whereas in protein, "N"=asparagine.

To mark the sequences in a set as being circular or linear.	`circular(sequence-set)`
	
	`linear(sequence-set)`
		`$A=circular(myfile.fasta)`
		
		`$B=linear(myfile.gb)`
		
		`myfile.gb=circular(myfile.fasta)("TAG",lend+4)`
		
		`myfile.gb=circular("ATG"+$B+"TAG")`
			This ability may be useful for sequences originating in formats where type is unspecified, such as FASTA files.
			
			If you use either of these functions, the information will be added to the output file, where allowed (e.g., *.gbk). The presence of sequence type information may affect the results of searching in an endpoint expression. For example, a match can cross the origin in a circular sequence, but not in a linear one.

Objective	Expression	Example	Comments

To collect multiple sequence-sets into one	`collect(sources, specified either as individual arguments or as single-quoted file patterns)`	`$a=collect(foo.fasta, bar.fasta, baz.fasta)`
		
		`$a=collect("KSLLQQLLTE", "ARTKQTAR", "RPKPLVDP")`
		
		`$a=collect('C:/MyFolder/*.fasta')`
		
		`collect('*.gb', '*.genbank')`
			This function accepts one or more arguments, each of which may be a sequence expression or a file pattern. A file pattern is a single-quoted search string that can match zero or more file paths. Wildcards (asterisks) may be used in the filename part of this string.
			
			This function allows:
			
			• Multi-sequence files to be defined within a script.
			• Multiple literals or files to be used in the function argument.
			• A custom subset of the sequences in a file, by collecting individual sequences within it.
			• A custom set to be defined once in a script and then re-used later.
			• Filename patterns to be combined so as to refer to filenames with multiple extensions (see lower-most example at left).
			
			Note that it is possible for a sequence to appear more than once in the resulting output.

<table>
<tr><th>Objective</th><th>Expression</th><th>Example</th><th>Comments</th></tr>
<tr><td rowspan="2">To sample the sequences in the input set</td><td><code>sample(sequence-set, argument=value)</code>

(See description of arguments, below)</td><td><code>sample("foo.fasta", from=10000, to=20000, by=10)</code>

<code>sample("foo.fasta", p='0.95')</code>

<code>sample("foo.fasta", name='GEK*')</code></td><td>This can be useful for separating reads into different sets, or for reducing a very large number of reads to a smaller number (e.g., because of software limitations).

Each of the arguments is optional. Any combination of arguments can be used, in any order. At most one of each argument may be used.

The output sequences are in the same order in which they appear in the original set.</td></tr>
<tr><td colspan="3">Arguments for the expression <code>'sample'</code>:
from - defines the inclusive lower bound of the included sequences.
to - defines the inclusive upper bound of the included sequences.
by - includes only every nth element.
p - is the probability that each element will be included. Calculations are made separately for each element. The elements chosen can differ between executions. The number of elements chosen is not deterministic. For large numbers of sequences it is likely close to [p * number of sequences].
name - specifies matching sequence names, with a single-quoted string that may include an asterisk (*) as a wildcard.
minLength - integral expression specifying the minimum length of matching sequences, inclusive.
maxLength - integral expression specifying the maximum length of matching sequences, inclusive.
contains - matches any sequence on either strand that contains the single-quoted value, which may contain ambiguity codes.</td></tr>
</table>

Actions

Miscellaneous actions that can be performed from within the SeqNinja shell include:

Objective	Expression	Examples	Comments

To separate sequences into multiple files	`split(file name, argument=value)`	`split(foo.fasta, to='*.fasta')`
		
		`split(foo.fasta, to='foo_*.fasta')`
		
		`split(foo.fasta, to='C:/data/foo_*.fasta')`
		
		`split(foo.fasta, to='*.fasta', limit=2000)`
			This action separates each sequence into its own file. The asterisk in the filename is replaced by the sequence name.
			
			Allowable arguments:
			
			to - (required) Specifies a filename pattern for the destination file paths. It must be a single-quoted string with an asterisk in the filename. The pattern may have a relative or absolute path. Prefixes and/or suffixes can be included in the filename.
			
			limit - (optional) Limits the number of sequence files that can be generated. Otherwise, accidental specification of a file containing millions of reads could stress some operating systems. The default limit is 1000 sequence files. Note that if `!conversion=full` is specified, each sequence file may be accompanied by auxiliary files.

To exit an interactive session of SeqNinja	`exit`
	
	`quit`
		`exit`
		
		`quit`	

Objective	Expression	Examples	Comments
To display information in the terminal or command shell	`print(arguments)`	`print('Sequence=', $myseq, 'length=', %len)`	Accepts an arbitrary number of arguments, each of which is printed to the standard output stream. An extra space is included between the output of each argument automatically, with the exception of single-quoted string literals. See the topic *Escape Codes* for information on how to use character escape codes together with `print`.

To add a timestamp	`timestamp(optional arguments)`	`timestamp('Sequence=', $myseq, 'length=',%len)`
		
		`timestamp`
			Same as `print`, but also adds a timestamp. Note that `timestamp` can also be invoked without any arguments, in which case it simply returns the current time.

To get help	`help`	`help`	Displays the SeqNinja help documentation.

Creating a Program to Use as MegAlign Pro Input

MegAlign Pro is Lasergene's application for performing multiple alignments of large DNA, RNA, or protein sequences using one of three alignment methods: MUSCLE, Mauve, or Clustal Omega. MegAlign Pro also offers several post-alignment analysis tools, including phylogenetic tree generation and access to a distance matrix, showing distances between each pair of sequences.

When you use MegAlign Pro, you need to enter the sequences you wish to align. It is possible to add a SeqNinja *.star project file in lieu of the usual sequence files. The advantage of adding a .star project to MegAlign Pro is that it will always call up the most up-to-date version of the input data for use in making its output. So if you change, add, or delete the input data that the .star file points to, the .star file will update this information in MegAlign Pro.

When you use a SeqNinja Project (*.star) file as input for MegAlign Pro, note that the project is read as a "set of sequences," and is not actually executed, meaning no sequence files are written. This type of project has limitation that any input files referenced in the program must already exist. In other words, the program should not reference an input file that doesn't exist until it is created earlier in the same program. Instead:

- Replace all the references to the input file with a variable (e.g., $a), or…

- Create a SeqNinja program to create the necessary files and run the program. Then, in the SeqNinja program you wish to add to MegAlign Pro, reference those already-created files.

When creating a SeqNinja program for use as MegAlign Pro input, you may break statements into multiple lines before or after the following operators:

,	comma
+	plus sign
=	equals sign
+=	plus equals sign

For examples, any of the following SeqNinja steps would be read as a set of sequences in MegAlign Pro:

Example 1:

```
$a = $b
  + $c
```

Example 2:

```
$a = $b +
  $c
```

Example 3:

```
$a = sample("myfile.fasta",
  from=100,
  to=200)
```

Example Scripts

The following scripts are intended only as a starting point, and must be adapted, as necessary, for your particular data and goals.

Insert a missing base into a sequence

Goal	To insert a "G" between 39352 & 39353 in an existing sequence.
Script	`newseg.gbk=oldseq.gbk(lend,39352)+"G"+oldseq.gbk (39353,rend)`

Reverse complement a sequence

Goal	To reverse complement (flip) 4750 reads in the file MID3_13.fas
Script	`"rcMID3_13.fas"=~("MID3_13.fas")`

Concatenate and de-concatenate contigs for gene finding

Goal 1	To merge contigs into a pseudo-molecule with a 36 bp linker that introduces a stop codon (linker: NNNNNCACACACTTAATTAATTAAGTGTGTGNNNNN) in all six frames.
Script 1	`$linker="NNNNNCACACACTTAATTAATTAAGTGTGTGNNNNN"` `pseudo.fas=contig00001.fas+$linker+contig00002.fas`
Goal 2	After annotation or gene-finding via pseudo.gbk, break the pseudo-molecule back into contigs: contig00001 length=116581 contig00002 length=97153
Script 2	`contig00001.gbk=pseudo.gbk(1,116581)` `contig00002.gbk=pseudo.gbk(116618,213770)`

Extract a specific set of genes or proteins from multiple genomes

Goal	First, to generate a multiple-genome file called "*all.gbk*" from all the .gbk files in the directory "genomes". Second, to extract a specific set of genes or proteins from "*all.gbk*."

Script	`!datapath:in = "/Users/[UserName]/Desktop/SN_test/genomes"`
	`!datapath:out = "/Users/[UserName]/Desktop/SN_test"`
	`"all.gbk" = collect( '*.gbk' )`
	`"MLST.fas" = extract(`
	`"/Users/UserName/Desktop/SN_test/all.gbk",`
	`'CDS:/gene="adk",CDS:/gene="fumC",CDS:/gene="gyrB",CDS:`
	`/gene="icd*",CDS:/gene="mdh",CDS:/gene="purA",CDS:/gene=`
	`"recA"' )`

Note: It is necessary to either have a different `datapath:in` and `datapath:out` at the point of the `collect` step, or to use the full path to *all.gbk*. Otherwise, the result will be an endless loop.

Convert one version of a sequence to another

The following scripts can be used to convert genome coordinates between assemblies or to migrate annotations between different versions of a genome while preserving annotations.

In this example, researchers identified seven mutations in *E. coli* strains that did not appear in the reference sequence. They wanted to determine the impact of those changes on the features annotated in the genome. Therefore, they needed to migrate annotations between different versions of a genome while simultaneously converting genome coordinates between assemblies.

Goal 1	To convert the reference sequence into the mutant sequences with full annotation. The following changes are required: • 1 bp SNP "A" to "G" • insert "G" (duplicate preceding "G") • 1 bp SNP "C" to "T"
Script 1	`Mutant_A.gbk=Reference_Seq.gbk(1,547693)+"G"+ Reference_Seq.gbk(547695,547832)+Reference_Seq.gbk (547832,3957956)+"T"+Reference_seq.gbk(3957958,rend)`
Goal 2	To convert the annotated reference sequence to the mutant-B sequence, making the following changes: • <IS1< insertion + 8 bp target duplication • 1 bp SNP "A" to "G" • insert "G" (duplicate preceding "G") • > IS5 > insertion + 4 bp target duplication • insert "CC" • 1 bp deletion/frameshift (-G) • 1 bp SNP "C" to "T"
Script 2	`Mutant_B.gbk=Reference_seq.gbk(1,257907)+complement (IS1.fas)+Reference_seq.gbk(257900,547693)+"G"+ Reference_seq.gbk(547695,547832)+Reference_seq.gbk (547832,1298721)+IS5.fas+Reference_seq.gbk (1298718,2171386)+"CC"+Reference_seq.gbk (2171387,3558477)+Reference_seq.gbk(3558479,3957956)+"T" +Reference_seq.gbk(3957958,rend)`

Divide a genome into pieces of a specified length with a specified overlap

Goal	To break up a genome into pieces of length N with overlap m, e.g., for simulations.
Script	`pieces.fas=cut(U00096.2.gbk,240,60)`
Output	25775 sequences of 240 bp, overlapping by 60 bp, plus the final part of the sequence.

Extract segments from a sequence

Goal	To extract the C-regions (exons), J-segments and V-segments from this T-receptor locus sequence (AF159056), producing output in FASTA format.
Script	`4Matt.fas=extract(AF159056.gbk,'V_segment,J_segment, C_region')`

Output (Matt.fas)	`> Matt:1`
	
	`Atgcggtgggccctagcggtgcttctagctttcctgtctcctggtgagtgcgctgc`
	`ctacagagaggatcacgggttttgttttgttttgttattttcttctttgcaagga`
	`gcgacatactaagaaatgcctcattatattttgtgttgttcccattgcagccagtc`
	`agatatcttccaacttggaagggagaacgaagtcagtcaccaggctgactgggtca`
	`tctgctgaaatcacctgtgatcttcctggagcaagtaccttatacatccactggta`
	`cctgcaccaggaggggaaggccccacagtgtcttctgtactatgaaccctactact`
	`ccagggttgtgctggaatcaggaatcactccaggaaagtatgacactggaagcaca`
	`aggagcaattggaatttgagactgcaaaatctaattaaaaatgattctgggttcta`
	`ttactgtgccacctgggacagg`
	
	`> Matt:2`
	
	`atgcagtgggccctagcggtgcttctagctttcctgtctcctggtgagtgcgctgc`
	`ctacagagaggatcacgggttttgttttatttcttctttgcaaggagtaccata`
	`ctaaggaattcctcattatattttgtgttgttcccactgcagccagtcagaaatct`
	`tccaacttggaagggagaacgaagtcagtcatcaggcagactgggtcatctgctga`
	`aatcacttgtgatcttgctgaaggaagtaacggctacatccactggtacctacacc`
	`aggaggggaaggccccacagcgtcttcagtactatgactcctacaactccaaggtt`
	…
	`> Matt:3-25(`similar; omitted here to conserve space)
	…
	`> Matt:26`
	
	`atacactactgctgcagctcacaaacacctctgcatattacatgtacctcctcctg`
	`ctcctcaagagtgtggtctattttgccatcatcacctgctgtctgcttggaagaac`
	`ggctttctgctgcaatggagagaaatca`

Extract annotated features from a genome

This example can be adapted to extract relevant annotated features (e.g., specific feature types or all features with specified annotations) for uses such as building BLAST databases or consensus matrices, or performing alignments.

Goal	To extract a set of annotated CDS features from a genome as protein sequences
Script A	`m54sCDS.gbk=extract(m54s.gbk, 'CDS')` `m54s_proteins1.fas=translate("m54sCDS.gbk")`
Output A Script A generates an overlapping CDS (specifically, the yeaC fragment). Using Scripts B or C solves this problem.	LOCUS U00096:yeaA 414 bp DNA 13-JAN-2012 FEATURES Location/Qualifiers Source 1..414 Source complement (1..414) /seqninja_feature_id="0000000001//Users/guy/Desktop/seqninja_testing /m54s.gbk:1" /note="***Needs review***Cut segment head by 1860039 and tail by 2778768 units." /organism="Escherichia coli" CDS 411..414 /gene="yeaC" /seqninja_feature_id="0000000012//Users/guy/Desktop/seqninja_testing /m54s.gbk:1" /note="***Needs review***Cut segment tail by 314 units." CDS 1..414 /gene="yeaA" /seqninja_feature_id="0000000013//Users/guy/Desktop/seqninja_testing /m54s.gbk:1" ORIGIN 1 atggctaata aaccttcggc agaagaactg aaaaaaaatt tgtccgagat gcagttttac 61 gtgacgcaga atcatgggac agaaccgcca tttacgggtc gtttactgca taacaagcgt 121 gacggcgtat atcactgttt gatctgcgat gccccgctgt ttcattccca aaccaagtat 181 gattccggct gtggctggcc cagtttctac gaaccggtaa gtgaagaatc cattcgttat 241 atcaaagact tgtcacatgg aatgcagcgc atagaaattc gttgcggtaa ctgtgatgcc…
Script B Scripts B and C produce identical results, but only B also outputs the nucleotide sequence file.	`m54sCDS.fas=extract(m54s.gbk, 'CDS')` `m54s_proteins2.fas=translate("m54sCDS.fas", '/transl_table=11')`
Script C	`m54s_proteins3.fas=translate("m54s.gbk")`

Add segments of literal sequence to every sequence in a set

Goal	To add "ATG" at the beginning and "TAG" at the end of every sequence in a multi-sequence file named *two.fasta*. The file *two.fasta* contains two sequences:
	`aaact`
	`aaactct`

Script	`print("ATG" + two.fasta + "TAG")`
Output	`ATGaaactTAG` `ATGaaactctTAG`

Collect sequences with certain attributes into a single file

Goal	The input directory *C:\data* contains the files *alpha.fas*, *alphabet.fasta*, *alpha.fap*, and *beta.fas*. The goal is to collect all the sequences in the directory whose sequence sets begin with "alpha," and whose extensions begin with ".fas," and write them to the file *results.fasta*.
Script	`!datapath = "C:\data"` `"results.fasta" = collect('alpha*.fas*')`
Output	Collects only the sequences from sequence sets *alpha.fas* and *alphabet.fasta* and saves them as a multi-sequence file named *results.fasta*.

Note: The output can be split back into multiple files using the procedure in the next topic, *Split a multi-sequence file into individual files*.

Split a multi-sequence file into individual files

Goal	To split a multi-sequence file into individual files according to specified criteria.
Script	`split("results.fasta", to='alpha_*.fas')`
Output	The two files (*alpha.fas* and *alphabet.fasta*) comprising the multi-sequence *results.fasta* file are saved as separate sequence files using the names in their definition lines, prepended by “alpha_”.

Appendix

Supported File Types

For a list of file formats supported by this DNASTAR product, please see the Supported File Types page of our website.

Research References

Schroeder JL and Blattner FR (1982). “Formal description of a DNA oriented computer language.” Nucleic Acids Res 10 (1):69-84 [PMID: 7063408; PMCID: PMC326115].