V-FAT USERS GUIDE

• General Description •

V-FAT (Viral Finishing and Annotation Toolkit) is a toolset to go from the raw contigs output of a standard *de novo* assembler to an oriented, frameshift-corrected and annotated assembly of the whole genome. It is a reference-guided process that is meant to improve the final result of a *de novo* assembler by providing additional information. If read data are provided, they will be used to improve the contig merging and the frameshift correction by looking for read support. It has been designed for use on viral RNA genomes, which are relatively short (10-15 kb) and free of repetition. It should work on any type of data that share these characteristics.

It has 5 main steps, which will be explained in more details in the section devoted to each script:

1) orientation and filtering of the raw contigs
2) merging of the contigs where they overlap based on a reference alignment
3) correction of frameshifts found in coding regions
4) annotation of the genes based on the new assembly
5) quality assessment, collecting statistics about the assembly, generating coverage plots and raising flags about potential issues in the final assembly.

This package includes the following scripts. A more detailed description of each along with their options will follow:

1) **vfat.pl**. This script is essentially a wrapper that runs through all of the 5 main scripts from start to finish.

2) **orientContig.pl**. This script orients the contigs to the reference and filters out contigs that are too distant to be believed to be from the same organism, or too short.

3) **contigMerger.pl**. This script merges contigs together that overlap when aligned to a reference genome. If read data are provided, the merger will look for read support when deciding which contig to favor if there are variants in the overlapping regions.

4) **fixFrameshifts.pl**. This script looks for frameshifts (regions with gaps that are not multiple of 3s) within coding regions and fixes them if there are reads supporting it. It is also possible to force correction in homopolymer regions even if no reads are provided.

5) **annotate.pl**. This script will use a database of peptides found on the reference genome and the external software genewise to locate genes on the assembly

and annotate their position. It will also look for potential issues in the genes such as remaining frameshifts, lack of an expected start or stop codon, etc.
6) **QA_stats.pl**. This script looks into the output files from the previous scripts in the pipeline as well as read alignments against the reference and the assembly to calculate statistics, generate coverage plots using R, and highlight potential issues in the genome.

7a) **runMosaik2.pl**. This a perl utility script to help running Mosaik with the various options required by the pipeline and to convert the output in the .qlx format, which is used by QA_stats.pl. The read alignments will also be output in the bam format.

7b) **samToQlx.pl**. This the perl utility script used to convert the Mosaik output in the .qlx format, which is used by QA_stats.pl.

8) **configPaths.pl**. This is a utility script meant to modify the paths to the various scripts and external software used by the pipeline in each part of the pipeline without having to manually open the files. See the end of the section for specific details on how to setup at the Broad.

The scripts **qlxToSam.pl**, **fastq2fasta.pl**, **fqpair2fasta.pl** and **translateDna.pl** are also included in the package for file conversion purpose. See the file format section for their use.

• Citing V-FAT •

*** Bioinformatics Application Note Reference incoming ***

• Required External Software •

1) **MUSCLE v3.8**. The MUSCLE aligner (http://www.drive5.com/muscle/) is used by the other scripts when global alignments are required. Using a different version of MUSCLE might require modifying command lines in the scripts, but should not be a major endeavor.

2) **R**. R is a statistical package that is used by V-Profiler in order to generate Heatmaps. It is available at http://www.r-project.org/. Version 2.9 or more is required.

3) **Mosaik**. Mosaik is a reference-guided assembler and short-read aligner using a Smith-Waterman based algorithm. The read alignment process is used by RC454. It can be found at http://bioinformatics.bc.edu/marthlab/Mosaik. The version used and tested extensively with these other tools is 2.1.

4) **GeneWise**. GeneWise compares a protein sequence to a genomic DNA sequence, allowing for introns and frameshifts. It is used to identify gene positions in a new assembly when comparing to a list of reference peptides. It can be downloaded at : ftp://ftp.ebi.ac.uk/pub/software/unix/wise2/. The version used and tested is wise2.2.0

• Quick Start •

While there is many parameters that can let you customize your analyses and all scripts included can be run individually, in most cases all you should need to do is do a quick installation/setup and run the main wrapper. Here is a step-by-step procedure of the minimum that is required to do:

1) Install all the required external softwares specified above if you do not have them. Mosaik is only required if you supply read data.

2) Fill in the paths in configfile.txt and run configPaths.pl. For more details, see the full section about the script. The command line for it is:
   perl configPaths.pl configfile.txt

3) If you do not already have them, make a gene list, amplicon positions list and peptides folder for the reference you want to use. See the "File Formats" section at the end for details. A folder containing multiple correctly formatted reference genomes is contained in the package (ViralReferenceGenomes) and has among others HIV, DENV 1 to 4 and WNV references ready.

4) Run vfat.pl wrapper with the following command line:

   perl vfat.pl -contigs <contigs.fa> [-readfa <reads.fa> -readq <reads.qual>] -ref <reference.fa> -genelist <ref_genelist.txt> -amps <ref_amplicons.txt> -pepfolder <peptidesFolder> -o <outputBasename>

   If you wonder what some of the files need to be or what the parameters available are, see the vfat.pl section as well as the sections covering the particular script you have questions about. Each section will also explain in details what the output files coming out of it are.

• Script Usage •

1) V-FAT:

vfat is the main wrapper that will run the whole process from start to end. It has 5 main steps, which will be explained in more details in the section devoted to each script:

1) orientation and filtering of the raw contigs
2) merging of the contigs where they overlap based on a reference alignment
3) correction of frameshifts found in coding regions
4) annotation of the genes based on the new assembly
5) quality assessment, collecting statistics about the assembly, generating coverage plots and raising flags about potential issues in the final assembly.

Algorithm overview :

The exact steps done by contig2assembly are the following:

1) run orientContig
2) run contigMerger
3) run fixFrameshifts
4) If reads are supplied, runMosaik, once against the reference and once against the fixed assembly
5) run annotate
6) run QA_stats
7) sort output files

Command Line

The detailed command line for vfat is the following:

vfat.pl -contigs <contigs.fa> [-readfa <reads.fa> -readq <reads.qual>] -ref <reference.fa> -genelist <ref_genelist.txt> -amps <ref_amplicons.txt> -pepfolder <peptidesFolder> -o <outputBasename>

If your reference data is organized in a specific way, you can also use the -virus option that will fill out most of input file parameters for you (see –virus parameter for more details):

vfat.pl -contigs <contigs.fa> -virus <virus> [-readfa <reads.fa> -readq <reads.qual>] -o <outputBasename>

Input Files:

-contigs <contigs.fa> : a fasta file containing all the contigs obtained from the assembler. The contigs will be renamed ‘contig_0’, ‘contig_1’, etc. The number corresponds to the order of the contig in your original file.

-ref <reference.fa> : a fasta file of the reference genome that will be used to orient the contigs.

-genelist <reference_genelist.txt> : Gene list with their start and stop positions on the reference (see “file format” section for more details)

-amps < reference_amplicons.txt > : Start/stop positions for the amplicons used in sequencing. The format used is the same as the genelist format (see “file format” section)

-pepfolder <peptides_folder> : Folder containing the peptides fasta used by GeneWise and a [Virus]_Peptides_Features.txt parameter files. See the “file format” section for more details

-readfa <reads.fa> : optional, reads in fasta format

-readq < reads.qual > : optional, read quality in spaced-delimited integers format

Parameters:

Only some parameters specific to the functioning of the V-FAT wrapper will be listed here. Note that ALL parameters from any of the 5 main scripts (orientContig, contigMerger, fixFrameshifts, annotate, QA_stats) can be given to V-FAT and it will use them when calling the program it applies to. For use of these parameters, see the section specific to each script.

The value between the brackets <> is the default value if the parameter is not entered. The type of the parameter is written in parenthesis (). A ‘boolean’ parameter does not need any value specified, you can simply do -parameter.

-sequencer <illumina> (string)
Type of sequencing data. This will modify the parameters used by the Mosaik aligner to better handle the known error modes of each type of sequencing. Currently supports ‘illumina’ and ‘454’ (others will use the illumina parameters).

-fakequals <0> (integer)
Fakes quality scores in the Mosaik alignments to a given score (integer set by the parameter). This will not affect the functionality of V-FAT and will speed it up, but the qlx files would not be proper for use with some other software like V-Phaser. In general this parameter is recommended.

-virus <NULL> (string)
Specify the virus to use for reference data (including gene list, amplicons and peptides folder). This option requires the reference files to be named and located in a specific manner. The virus name is case sensitive. If you want to use this parameter, the files need to be named and organized as follow:

The reference fasta name must be : [virus]_Reference.fasta
The genelist name must be : [virus] _Reference_genelist.txt
The amplicons list name must be : [virus] _Reference_amplicons.txt
The peptide folder name must be : [virus]_Peptides (see File Formats section for details about the Peptide folder)

The script will look for the files in the location specified by the variable refDataPath (set by the configPaths.pl script, see this section for details), followed by a folder named [virus]. An example of how files could be named and organized with data for 2 viruses, HIV and RSV:

```
refDataPath (as configured by configPaths.pl) : /my/home/folder/

HIV files:

/my/home/folder/HIV/HIV_Reference.fasta
/my/home/folder/HIV/HIV_Reference_genelist.txt
/my/home/folder/HIV/HIV_Reference_amplicons.txt
/my/home/folder/HIV/HIV_Peptides

RSV files:

/my/home/folder/RSV/RSV_Reference.fasta
/my/home/folder/RSV/RSV_Reference_genelist.txt
/my/home/folder/RSV/RSV_Reference_amplicons.txt
/my/home/folder/RSV/RSV_Peptides
```

The file format section has some extra details on the formats of each individual file.

The package already includes a folder named ViralReferenceGenomes that contains correctly formatted reference data for multiple viruses such as HIV, DENV 1 to 4, and WNV (among others). Adding the path to this folder in the configfile.txt will allow an easy use of the -virus option.

-details <NULL> (boolean)
Using this parameter will keep all the temporary files that could be useful for debugging purposes or to manually modify part of the results (for example editing manually a frameshift). All of these additional files will be in the folder <output>_additionalFiles. In the output files section for each script, the files that are under “secondary outputs” will be moved to this folder and deleted unless this

parameter is set. Occasionally some of the other outputs will too (for example the assembly generated by contigMerger will be moved, because it is replaced by the one returned by fixFrameshifts)

<u>**Output Files:**</u>

a) <output>_commandlog.txt : File containing all the command lines sent to the terminal by the wrapper with detailed parameters and input file names. The main purpose of this file is to allow you to rerun specific steps of the analysis easily, with different parameters if required, without having to go through the whole wrapper again. It can also allow you to manually modify a specific file (manually fix some frameshifts that were reported but not automatically corrected by the fixFrameshifts script for example) and then run the following steps using this modified file instead.

Secondary outputs:

b) <output>_additionalFiles : V-FAT will move in this folder output files that could be useful but are considered secondary in interest to prevent an excess of files in your main results folder. This folder will be deleted unless the parameter -details is specified.

All other output files are part of the specific outputs of each individual script and will be detailed in each individual section.

2) orientContig*:*

orientContig.pl is a script that orients raw contigs obtained from an assembler to a reference genome, and filters out contigs that are not considered good enough to be used in merging. The filtering is done based on 2 parameters :

a) Length of the contig. All contigs below a set minimum length are filtered out.

b) Quality of the alignment of the contig to the reference.

The orientation of the contig is determined based on both the percentage of gaps and the longest segment of the contig aligned to the reference without containing a gap longer than a set maximum length.

Algorithm overview :

For each contig in the input file:

1) Check if contig passes basic length filter

2) Align contig to reference in both orientations

3) Calculate percentage of gap in both orientations

4) Calculate longest aligned segment for the contig. Segments start when you have data both for the reference and the contig, and end when there is a gap of length greater than [maxorigaplen]. Contigs where the longest continuous segment is shorter than [minlongcont] in both orientations are filtered out as they are more likely to be from a different organism or misassembled.

5) Compare the gap % and longest contig value in both orientations in order to determine the correct orientation. Length of the longest segment is the most important parameter, but if its difference between both orientations is less than 2-fold, gap percentage will be used to help determine which is correct.

In the vast majority of cases, there is a very clear difference between both orientation and no ambiguity.

Command Line

The main command-line for running orientContig is:

orientContig.pl <contigs.fa> <reference.fa> <output>

Input Files:

<contigs.fa> : a fasta file containing all the contigs obtained from the assembler. The contig headers should not contain spaces or special characters.

<reference.fa> : a fasta file of the reference genome that will be used to orient the contigs.

Parameters:

3 parameters can be set (with their default value) :

-mincontlen <350> (integer)
Minimum length of a raw contig. Any contig below this length will be filtered out from the start.

-minlongcont <100> (integer)
Minimum length of a segment from a contig that must align to the reference without a gap longer than [maxorigaplen] for the contig to be considered.

-maxorigaplen <10> (integer)
Maximum length of a gap allowed within a segment of a contig when determining the longest contig.

Output Files:

a) <output>.fa
Fasta file containing all the remaining contigs in the correct orientation

Secondary outputs:

b) <output>_<contigname>_orientalign.afa
Aligned fasta file containing the contig <contigname> aligned to the reference in the proper orientation. Only present for contigs that passed filters.

3) contigMerger :

contigMerger.pl is a script that takes a list of contigs aligned to a reference and merges them where they overlap. If no overlap are present, it will fill the assembly with strings of Ns. If read data is given, it will use it in order to determine which variants have more read support when overlapping contigs have differences. If no read data is given, the merging will occur in the middle-point of the overlapping region.

Algorithm overview :

1) Split contigs into segments that have a continuous alignment to the reference. Segments start at a position where both the reference and the contig have a base aligning, and will extend until either reaching the end of the contig or until a deletion in the alignment longer than [maxseggap] or an insertion longer than [maxsegins]. More than one segment can be built per contig. Segments will be kept for merging purpose if they are longer than [minseglen] and do not have an internal fraction of gaps greater than [maxsegdel].
2) Map where the segments from each contig cover the reference and overlap with each other.
3) Add contigs to the list of contigs to merge in decreasing order of length until the segments from these contigs cover the reference (or as much of it possible given the available segments mapping)*. If reads are supplied, more contigs can be added until the reference is covered at [contigcov]X coverage. The reason to add these extra contigs is that they might not be the longest but could have higher read support, making them more representative of the underlying population.*
4) *If reads are supplied, align reads to each of the contigs in the list of potential contigs to use for merging.*
5) Merging process WITHOUT read support:
   a) Walk across the reference from start to finish
   b) When a single contig segment covers a stretch of the reference, add it to the merged assembly
   c) If multiple segments cover a stretch of the reference, add the one that ends at the latest position in the reference. If this segment is different than the one that was covering the previous stretch of the reference, merge them at the middle-point of their overlap.
   d) If no segment cover a certain stretch of the reference but you have segments covering the reference before and after, add a string of Ns covering this stretch of the reference to the merged assembly.
6) Merging process WITH read support:
   a) Walk across the reference from start to finish
   b) When a single contig segment covers a stretch of the reference, add it to the merged assembly

c) If multiple segments cover a stretch of the reference, for each difference that they have add to the merged contig the variant that has the highest read support.
d) If no segment cover a certain stretch of the reference but you have segments covering the reference before and after, add a string of Ns covering this stretch of the reference to the merged assembly.

Command Line :

The basic command line for running contigMerger is:

```
perl contigMerger.pl <orientContigOutput> <reference.fa> <output> [-readfa
<reads.fa> -readq <reads.qual>]
```

Input File:

<orientContigOutput> : base name of the output files of the orientContig script. It is important to have the same output name as the one entered in the orientContig command line because contigMerger will use all the output files from orientContig such as the contig to reference alignment to prevent rerunning them all again.

<reference.fa> : reference fasta file

<reads.fa> : optional input file, fasta of the reads

<reads.qual> : optional input file, quality score of the reads (must be given if the reads fasta is given)

Parameters :

-readfa <NULL> (string)
Name of the reads.fasta input file

-readq <NULL> (string)
Name of the reads.qual input file

-mincontlen <350> (integer)
Minimum length of a raw contig. Any contig below this length will be filtered out from the start.

-maxseggap <30> (integer)
Maximum gap length before splitting segments

-maxsegins <60> (integer)
Minimum length of a raw contig. Any contig below this length will be filtered out from the start.

-minseglen <50> (integer)
Minimum length of a contig segment that will be added to the merging process

-maxsegdel <0.25> (integer)
Maximum fraction of internal gap that a contig segment can have to be added to the merging process

-longcont <1500> (integer)
All contigs above [longcont] will be considered in the merging process, even if all the reference is already covered by longer contigs

-contigcov <2> (integer)
Fold-coverage of the reference after which no other contigs will be considered for merging (with the exception of contigs longer than [longcont])

-allcont <NULL> (boolean)
Forces the use of all contigs in the merging process regardless of coverage or length.

-sequencer <illumina> (string)
Type of sequencing data. This will modify the parameters used by the Mosaik aligner to better handle the known error modes of each type of sequencing. Currently supports 'illumina' and '454' (others will use the illumina parameters).

-fakequals <0> (integer)
Fakes quality scores in the Mosaik alignment to a given score (set by the parameter). This will not affect the functionality of contigMerger and will speed it up, but the qlx files would not be proper for use with some other softwares like V-Phaser. In general this parameter is recommended.

<u>Output Files:</u>

a) <output>_assembly.fa
Fasta file containing an assembly of the merged contigs

b) <output>_contigsMap.[pdf/R]
Pdf graph showing where the contigs that were used for the contig merging map on the reference genome. R file that was used to generate the graph is also conserved for manual modifications if desired.

c) <output>_largeDeletions.txt
List of the large deletions (deletions longer than [maxseggap]) found in the contigs that were used for the merging. Sometimes these large deletions can be filled by another contig in the merged genome but can represent real variants in the population.

d) <output>_mergingList.txt
Text file containing a list of the contigs that were merged together across each stretch of the reference. Each line contains the start and stop position of this stretch of the reference, then which contig was used to cover it (it could be a single contig, or a mix of contig if they overlapped and part of both were used, or a stretch of Ns if no contig was covering the region), and the start-stop positions in this contig. It also notes how many different contigs were necessary to build the assembly.

Example:

```
1     468   contig_2a 1-469
469   589   contig_2a 470-590
590   709   contig_4 175-294
710   856   contig_4 295-441
857   904   contig_4 442-489
905   952   contig_1 136-183
953   4096  contig_1 184-3324
4097  4219  123N
4220  7328  contig_0 98-3230
7329  7375  contig_0 3231-3277
7376  7421  contig_3 136-181
7422  7946  contig_3 182-706
7947  8039  contig_3 707-800
8040  8132  contig_6 95-187
8133  8240  contig_6 188-295
8241  8309  contig_6 296-364
8310  8378  contig_2b 176-244
8379  8764  contig_2b 245-960

*****************
NB CONTIG USED:   7

contig_0
contig_1
contig_2a
contig_2b
contig_3
contig_4
contig_6
```

Secondary Outputs:

d) <output>_vsRef.[mfa/afa]
Alignment of the merged assembly against the reference (.mfa is unaligned, .afa is aligned)

e) <output>_<contig>.[fa/qlx] (*only if read data is supplied*)
Fasta file of the contigs used for the merging, and the read alignment in qlx format against each of them.

4) fixFrameshifts :

fixFrameshifts.pl is a script that attempts to correct all the frameshifts found in the coding regions of the merged assembly. If a read alignment is supplied, it will look for reads supporting a modification of the assembly that would remove the frameshift. If no read alignment is supplied, it will make notes of all the frameshifts found, and optionally will correct frameshifts located in an homopolymer region.

Algorithm overview :

1) Align the merged assembly to the reference
2) Find all frameshifted windows in coding regions (which are based on a supplied gene list from the reference). A frameshifted window is a stretch of DNA of length ([readwindow] * 2 + length(central indel)) where you have a total of (inserted bases minus deleted bases) that is not a multiple of 3.
3) For each frameshifted window, determine if the central indel is in an homopolymer or not
4) Merge frameshifted windows that are closer than ([readwindow] * 2) nucleotides apart together
5) If a read alignment is supplied, correct frameshifted windows where the dominant window found in the reads does not contain a frameshift
6) As options, you can also : force correction of any frameshift in an homopolymer by extending/reducing the homopolymer length accordingly. This requires at least 1 read support if a read alignment is supplied; force correction by selecting the most frequent non-dominant window that corrects the frameshift as long as it's present in the reads in a fraction over [forcefix]; not fix any frameshift, only note them in the log.

Command Line :

The basic command line for running fixFrameshifts is:

```
perl fixFrameshifts.pl –fa <merged_assembly.fa> -ref <reference.fa> -genelist
<reference_genelist.txt> -o <output> [-qlx <readalignment.qlx>]
```

Input File:

-fa <merged_assembly.fa> : Merged assembly obtained by contigMerger. This could also be any assembly that you want to look at frameshifts on obtained by another method.

-ref <reference.fa> : reference fasta file

-genelist <reference_genelist.txt> : Gene list with their start and stop positions on the reference (see "file format" section for more details). Note that the software assumes that the reference gene annotations are correct and free of frameshifting indels.

-qlx <readalignment.qlx> : optional input file that will use read alignment to support the correction of frameshifts in the assembly. contigMerger, if used with supplied reads, will generate this file in its outputs.

<u>Parameters :</u>

-minhomosize <3> (integer)
Minimum length of an homopolymer to call the variant in an homopolymer. Suggested lengths are 3 for illumina data and 2 for 454 data.

-readwindow <5> (integer)
Length of the nucleotides window on each side of an indel to look for further indels that could remove the frameshift. It will also determine the size of the windows to look for in the reads that could correct the frameshifts.

-nofix <NULL> (boolean)
Adding this parameter will disable any automatic modification to the assembly. It will still note all the frameshifts and suggested replacement windows in the log. This parameter will take precedence over the -fixhomo and -forcefix parameters.

-fixhomo <NULL> (boolean)
This parameter will force the correction of frameshifts in homopolymers. If a read alignment is supplied, it will still require at least 1 read to support the correction.

-forcefix <0> (float)
If this parameter is set to a fractional value]0-1], it will correct frameshifted windows in the assembly even if they are the dominant window found in the reads, as long as the most frequent window that would remove the frameshift is found at a frequency >= <forcefix>.

<u>Output Files:</u>

a) <output>_fixed_assembly.fa
Fasta file containing the assembly with the frameshift fixes (if any) in it.

b) <output>_frameshifts_log.txt
Text file that keeps a lot of all the frameshift found in the genes. It tells you the positions in the assembly where the frameshifts happen, if it is in an homopolymer or not and the length difference with the reference (a negative number means that it is shorter than the reference, a positive number that it is longer). It will also show you the window around the frameshift in the assembly, and if you supply reads it will find the windows in the reads covering the same positions and the frequency of the top 3. It will tell you which window (if any)

could fix the frameshift, and if it did fix it or not in the fixed_assembly.fa file (depending on the parameters that you gave it). Here is an example of an entry in the log:

```
**********************************
Potential FS Window starting at 3064 in gene NS1 with gap of length -1
It contains 1 frameshifts:
3069 in homopolymer

Assembly Window :
GAAGTTAAAGC

Top 3 Read Windows:
GAAGTTAAAAGC          290
GAAGTTAAAGC           31
GAAGTTAAAAAGC         17

Best Replacement Window : GAAGTTAAAAGC (290/374)
FIXED

**********************************
```

Secondary outputs:

c) <output>_alignPair.[mfa/afa]
Alignment of the assembly to the reference before fixes

d) <output>_alignPair_fixed.[mfa/afa]
Alignment of the assembly to the reference after fixes

5) annotate :

annotate.pl is a script that takes fastas of peptides from the genes found on the reference and uses GeneWise to position them on the assembly. It will also return flags to point out genes that could still contain frameshifts, or have a short length, are missing a start or stop codon, etc. One of its outputs is in a format used for NCBI submission.

Algorithm overview :

1) Run GeneWise to locate the peptides on the assembly
2) Attempt to improve GeneWise results if they ignored stop codons or cut genes too early due to a mismatch near the ends.
3) Determine which genes are present, partial or missing. Note that GeneWise will always attempt to match a peptide on the assembly even if the alignment is very bad and will usually not return that a peptide is missing, so some parameters are required to differentiate a partial gene from a missing gene.
4) Verify if all flags are passed and return various output files

Command Line :

The basic command line for running annotate is:

```
perl annotate.pl –fa <fixed_assembly.fa> -ref <reference.fa> -genelist
<reference_genelist.txt> -pepfolder <peptides_folder> -o <output> [-align
<assemblyVsReferenceAlignment.afa>]
```

Note : before running, be sure to set the environment variable used by Genewise
setenv WISECONFIGDIR Path/To/Genewise/wise2.2.0/wisecfg/

Input File:

-fa <fixed_assembly.fa> : Final assembly obtained by fixFrameshifts.pl (and manually edited if required).

-ref <reference.fa> : reference fasta file

-genelist <reference_genelist.txt> : Gene list with their start and stop positions on the reference (see “file format” section for more details)

-pepfolder <peptides_folder> : Folder containing the peptides fasta used by GeneWise and a [Virus]_Peptides_Features.txt parameter files. See the “file format” section for more details

-align < assemblyVsReferenceAlignment.afa > : optional aligned fasta of the assembly against the reference. If it's not supplied, annotate will use Muscle to generate one.

Parameters :

-maxannogaplen <50> (integer)
Maximum gap length allowed in a gene before forcing the split into 2 exons.

-minpctid <0.4> (float)
Parameter to distinguish partial genes from missing genes. A partial gene must have an identity of at least 40% with the peptide sequence.

-minpctlen <0.2> (float)
Parameter to distinguish partial genes from missing genes. A partial gene must have a length of at least 20% of the peptide sequence.

Output Files:

a) <output>_annotation_genelist.txt
Text file containing the gene names and their start/stop positions in the assembly in tabulated format. Example:

```
Caps  87    428
Memb  429   926
…
```

b) <output>_annotation_ncbi.txt
Text file containing the genes, their product and their start/stop positions in ncbi-submission format. It also adds a 5'UTR and 3'UTR to the genome automatically before the first gene and after the last gene (unless the first/last gene is partial or missing). Example:

```
1      86    5'UTR
                   note  indels in UTR have not been validated
87     428   mat_peptide
                   product     anchored capsid protein
429    926   mat_peptide
                   product     membrane glycoprotein precursor
```

c) <output>_ annotation_summary.txt
Text file containing the gene names, start/stop positions, as well as various flags that can identify potential problems in the genes. The flags are:

Completion: Is the gene complete? Can be "Full", "Partial" or "Missing".
Size : Is the gene within 10% length of the reference one? 1 = yes, 0 = no
Start/Stop : Does the gene have expected start/stop codon? 1 = yes, 0 = no

** Note that the requirement of a start/stop codon is stated in the Peptides_Features.txt input file. It will not require genes to have a start/stop codon to pass the flag unless specified in the input file*.
Frameshift : Does the gene contain a frameshift? 1 = no, 0 = yes
Internal N string : Does the gene contain an internal string of N (which would mean it was artificially bridged but was lacking data)? 1 = no, 0 = yes
Exons : Does the gene contain the expected number of exons? 1 = yes, 0 = no
** Expected number of exons comes from the reference genelist. Genes will be expected to have 1 exon, unless gene names are followed by _exon#, in which case the number of exons will be calculated.*
All : Are all flags passed (i.e. = to 1 or complete)? 1 = yes, 0 = no

Example:
```
Gene  NbExon      Completion  Start Stop  Flag Size   Flag Start/Stop
      Flag Frameshift   Flag Internal N String  Flag Exons  All Flags
Ok
Caps  1     Full  87    428   1     1     1     1     1     1
Memb  1     Full  429   926   1     1     1     1     1     1
```

Secondary outputs:

d) <output>_ annotation
Folder containing the raw results from GeneWise for each gene. See GeneWise documentation if you need more clarifications.

e) <output>_ annotation_aligned.[mfa/afa]
Aligned assembly to the reference.

6) QA_stats :

QA_stats.pl is a Quality Assessment script that summarizes various statistics about the assembled genome that was generated from the 4 previous steps. If read data is supplied, it will also calculate average coverage over the assembly, by gene and by amplicons, as well as provide coverage plots.

Algorithm overview :

If there are no reads supplied, there is no algorithm to speak of. The script only reads in input files and summarizes information already collected in the other steps.

If there is reads, there is a small algorithm used for calculating coverages:

1) Build nucleotide frequency tables at each position by counting the occurrence of each base at each position in the read alignments
2) Calculate the average coverage for each feature (full assembly, gene or amplicon) by taking the average of all counts per position covered by the feature in the read alignment against the reference.
3) Calculate the percentage of non-dominant mismatches and indels by counting each position where the base (or indel) that is the most present in the read is not the one present in the assembly.

Command Line :

The basic command line for running annotate is:

```
perl QA_stats.pl -assem <fixed_assembly.fa> -ref <reference.fa> -genelist
<reference_genelist.txt>  -amps <reference_amplicons.txt> -annot
<annotation_summary.txt> -mergelist <contigMerger_mergelist.txt> -mergeR
<contigMerger_contigMaps.R> -virus <virusName> -o <output> [-readfa
<reads.fa> -qlxref <readsToReferenceAlignment.qlx> -qlxass
<readsToReferenceAlignment.qlx>]]
```

Input File:

-assem <fixed_assembly.fa> : Final assembly obtained by fixFrameshifts.pl (and manually edited if required).

-ref <reference.fa> : reference fasta file

-genelist <reference_genelist.txt> : Gene list with their start and stop positions on the reference (see “file format” section for more details)

-mergelist < contigMerger_mergelist.txt > : <output>_mergingList.txt output file from the contigMerger script.

-amps < reference_amplicons.txt > : Start/stop positions for the amplicons used in sequencing. The format used is the same as the genelist format (see “file format” section)

-annot < annotation_summary.txt > : <output>_annotation_summary.txt output file from the annotate script.

-mergeR < contigMerger_contigMaps.R > : <output>_contigMaps.R output file from the contigMerger script

-virusName < virusName > : Name of the virus. If none is supplied, the script will assume that whatever is found in the reference file name before the first ‘_’ character is the virus name (example : HIV_Reference.fa). This is only used in some of the output texts and has no effect on functionality.

-readfa < reads.fa > : optional reads fasta file

-qlxref < readsToReferenceAlignment.qlx > : optional reads alignment to the reference in qlx format

-qlxass < readsToAssemblyAlignment.qlx > : optional reads alignment to the assembly in qlx format

Parameters :

None

Output Files:

a) <output>_QA_ StatsDetailed.txt
Output file that contains all the stats calculated by the QA script. Here is an example of an output file:

```
Files used:
Reference File : Path/To/Ref/DENV2_Reference.fasta
Assembly File : TestDen2_fixed_assembly.fa
Read Alignment vs Reference : TestDen2_alignVsRef.qlx
Read Alignment vs Assembly : TestDen2_alignVsAssembly.qlx
Reference Genelist : Path/To/Ref/DENV2_Reference_genelist.txt
Reference Amplicons : Path/To/Ref/DENV2_Reference_amplicons.txt
Contig merging list: TestDen2_merger_mergingList.txt
Contig merging R file: TestDen2_merger_contigsMap.R
Annotation summary : TestDen2_annotation_summary.txt
Reads file : Path/To/Reads/TestDen2_reads.fa

---------------------------
```

```
Reference data

Virus : DENV2
Length reference : 10723
Nb Amplicons : 4
Nb Genes : 11
Target region : 97-10272

---------------------------

Assembly QC General Stats

Length Assembly : 10668
% Reference Covered : 100.000%
Nb Contigs Used in Assembly : 2
Nb N strings inserted to merge Assembly: 0
Nb Genes Fully Covered : 11
Nb Genes Partially Covered : 0
Nb Genes Missing : 0
Nb Full Genes with Frameshift : 1
Nb Full Genes missing Start/Stop codon : 0
Nb Full Genes with N-string : 0

---------------------------

Assembly QC Coverage Stats

Nb Reads in input : 9852
% Reads Aligned to Reference : 76.421%
% Reads Aligned to Assembly : 76.726%

* Coverage Data vs Reference *
Average Coverage Target: 251.228
Standard Deviation Coverage: 108.472
%Reference Covered at 1X : 100.000%
%Reference Covered at 10X : 100.000%
%Reference Covered at 50X : 95.715%
%Reference Covered at 200X : 71.570%

* Coverage Data By Gene (vs Reference) *
Caps  56.579
Memb  217.669
Env   306.339
NS1   348.023
NS2A  236.627
NS2B  227.505
NS3   309.976
NS4A  330.118
2KPep 271.493
NS4B  298.539
NS5   155.945

* Coverage Data By Amplicon (vs Reference, non-overlapping region only)
*
Amp1  245.033
Amp2  211.811
Amp3  300.993
Amp4  70.104
```

```
* Coverage Data vs Assembly *
Average Coverage Assembly: 241.076
Standard Deviation Coverage: 117.546
% Positions with non-dominant base call : 0.000281214848143982
% Positions with non-dominant deletion call :   0
% Positions with non-dominant insertion call :  0
```

b) <output>_QA_ StatsSummary.xls
Contains most of the same information as the StatsDetailed.txt output, but in single row, tabulated format that makes it easy to concatenate in an Excel file.

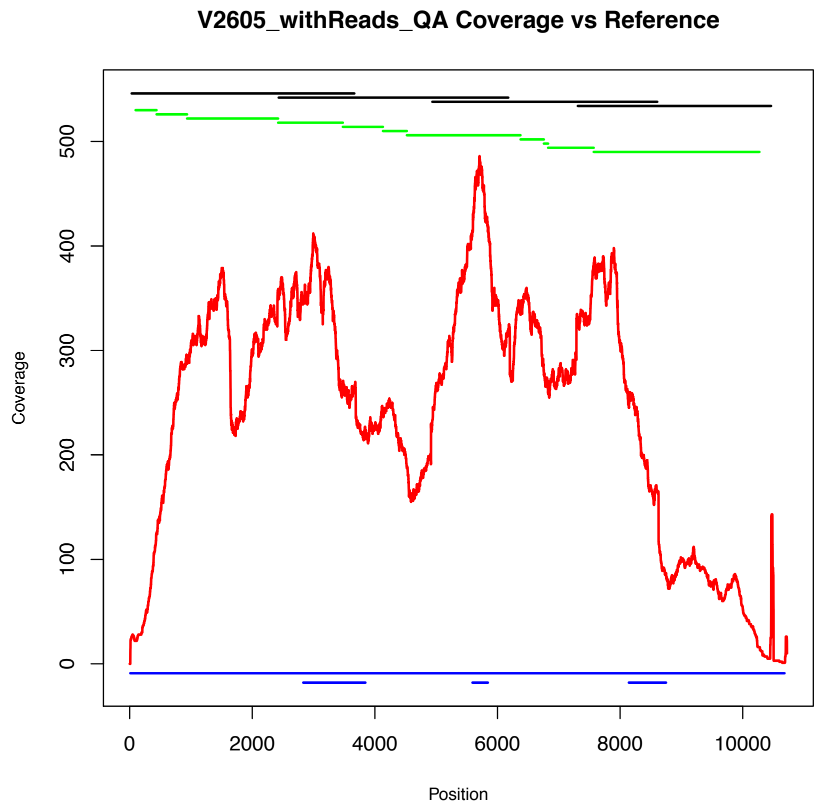
c) <output>_QA_ntfreq_ref.txt
Nucleotide frequency table for reads aligned against the reference. It’s tab-delimited and contains for each position the coverage and the frequency of each base, deletion or insertion. This is a raw nucleotide frequency table without any filtering or validation of variants.
Example:

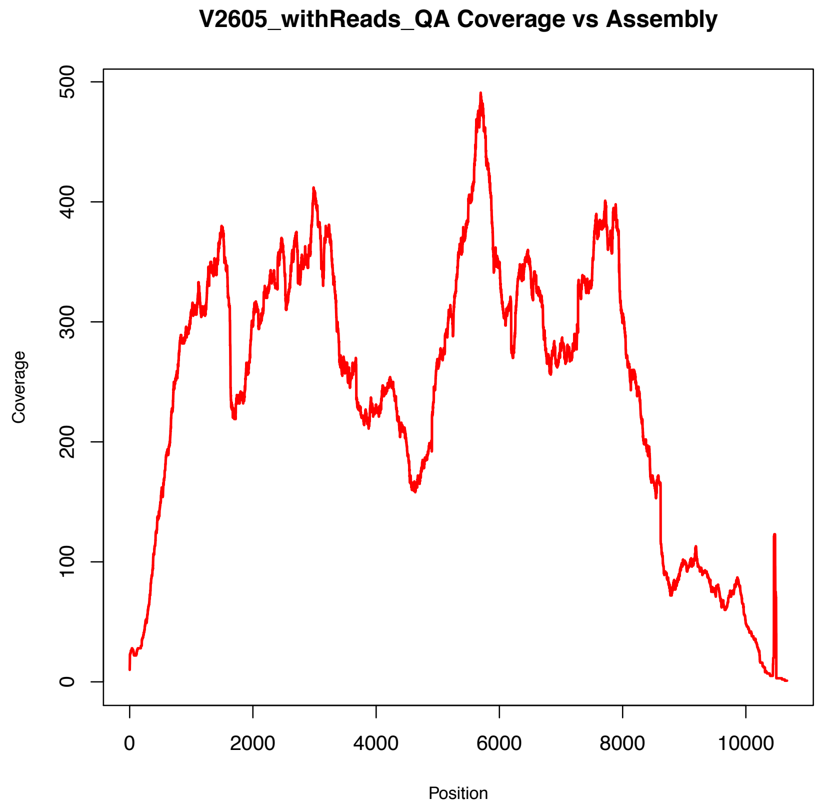
```
Pos   ConsensusNt Coverage    FreqA FreqT FreqG FreqC FreqDel
      FreqInsertion     Insertions(Count)
70    G     26    0     0     1     0     0
71    T     25    0     0.96  0.04  0     0
…
```

d) <output>_QA_ntfreq_assem.txt
Nucleotide frequency table for reads aligned against the assembly, same format as the reference one.

e) <output>_QA_coverageVsRef.pdf
Coverage plot against the reference. The positions are based on the reference, the red line is the coverage, the black horizontal lines are the amplicons, the green horizontal lines are the genes and the blue horizontal lines are the contigs. Example next page.

f) <output>_QA_coverageVsAssembly.pdf
Coverage plot against the assembly. The positions are based on the assembly, not the reference and so they might not match between both maps. Only the red coverage line is shown on this map.
Example next page.


V2605_withReads_QA Coverage vs Reference
Coverage
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
Position


V2605_withReads_QA Coverage vs Assembly
Coverage
0
100
200
300
400
500
0
2000
4000
6000
8000
10000
Position

7 a) runMosaik2

runMosaik2.pl is a utility script that serves 2 purposes : the first is to act as a wrapper script for MosaikBuild and MosaikAligner. It can also call the samToQlx.pl script if the -qlx parameter is set to generate the .qlx file format that is used by contigMerger, fixFrameshifts and QA_Stats from the .sam file format that is an output of Mosaik.

Command Line :

runMosaik2 can take multiple input formats. It can function either with reads in fasta and qual formats (often returned by 454 sequencing), paired or not, and in fastq format (often returned by illumina sequencing), paired or not. In the command line, include either -fa/-qual, -fa/-fa2/-qual/-qual2, -fq or -fq/-fq2.
The main command line for runMosaik2.pl is:

perl runMosaik2.pl -fa <reads.fasta> -qual <reads.qual> -ref <assembly/reference.fasta> -o <output>

Input Files:

-fa <reads.fasta> : reads in fasta format
-qual <reads.qual> : reads quality scores in spaced-delimited integers format

or

-fa <reads.fasta> : first mates in paired reads in fasta format
-qual <reads.qual> : first mates in paired reads quality scores in spaced-delimited integers format
-fa2 <reads2.fasta> : second mates in paired reads in fasta format
-qual2 <reads2.qual> : second mates in paired reads quality scores in spaced-delimited integers format

or

-fq <reads.fastq> : reads in fastq format

or

-fq <reads.fastq> : first mates in paired reads in fastq format
-fq2 <reads2.fastq> : second mates in paired reads in fastq format

-ref <assembly/reference.fasta> : assembly or reference that you want to align against, in fasta format

Options:

-hs <10> (integer)
Hash size for Mosaik alignment. See the documentation for more details.

-act <15> (integer)
Alignment candidate threshold for Mosaik alignment. See the documentation for more details.

-mmp <0.25> (float)
Maximum percentage of the read length that can be errors

-minp <0.25> (float)
Minimum percentage of a read that has to be aligned to keep it

-ms <10> (float)
Match score of the Smith-Waterman algorithm.

-mms <-9> (float)
Mismatch penalty of the Smith-Waterman algorithm.

-hgop <20> (float)
Penalty for opening a gap in an homopolymer for the SW algorithm

-gop <40> (float)
Gap opening penalty for the SW algorithm

-gep <10> (float)
Gap extending penalty for the SW algorithm

-nqsmq <20> (integer)
Minimum quality required for a base to pass the NQS filter

-nqsaq <15> (integer)
Minimum quality required for the neighborhood bases to pass NQS

-nqssize <5> (integer)
Size of the neighborhood on each side of the central base that is considered for the NQS filter

-bw <29> (integer)
Uses the banded Smith-Waterman algorithm. This greatly increases alignment speed, but seems to slightly reduce accuracy in highly diverse samples. Use for Illumina is recommended.

-nqvalue <1> (integer)
Q-Score given to a N that gets added by the script in the quality file. Default is 1. The important part here is that all bases that have the nqvalue will get ignored when looking at the neighborhood of the base in the NQS filter. This is to prevent an inserted N from ‘flagging down’ all adjacent bases. The score 1 is chosen by default because usually no base gets assigned a score this low.

-m <unique> (string)
Only keeps uniquely aligned reads. Use ‘all’ to include reads aligning in multiple locations.

-st <illumina>
Sequencing technology. Can also be ‘454’. See Mosaik manual for more details.

-qlx <NULL> (boolean)
Convert the sam output in the .qlx format.

-qlxonly <NULL> (boolean)
Only keeps the .qlx output and not the .bam or .sam

-fakequals <0> (integer)
If set to a positive integer number, will fake the quality of all nucleotides to the given value. This speeds up the creation of the .qlx file greatly if you do not care about the quality scores.

-mfl <600> (integer)
Medium fragment length. Only necessary for paired reads.

-param454 <NULL> (boolean)
Sets the following parameters to a value that is suitable to 454:
-gop <15>
-hgop <4>
-gep <6.66>
-st <454>

-paramillu <NULL> (boolean)
Sets the following parameters to a value that is suitable to illumina:
-gop <40>
-hgop <20>
-gep <10>

-annpe <NULL> (string)
Network file. This file is actually included in the Mosaik distribution. It should be located in the networkFile folder. For Mosaik 2.1.26, this file is named: 2.1.26.pe.100.0065.ann

-annse <NULL> (string)
Network file. This file is actually included in the Mosaik distribution. It should be located in the networkFile folder. For Mosaik 2.1.26, this file is named: 2.1.26.se.100.005.ann

Outputs:

<output>.bam : bam format output file

<output>.sam : sam format output file

<output>.qlx : .qlx alignment file (see File Format for details). If the -qlx parameter is set.

7 b) samToQlx

samToQlx is a script used by runMosaik2 to convert its bam output to a qlx format. The script can also be used on its own to convert from bam or sam to qlx.

Command Line :

perl samToQlx.pl <sam/bam input> <reference.fasta> <output>

Input Files:

<sam/bam input> : sam or bam input file. If using a bam input, set the bam parameter.
<reference.fasta> : reference fasta file

Options:

-bam <NULL> (boolean)
Output file is in bam format instead of sam format. This will convert the bam file to a sorted sam file, and then convert to qlx.

-nqsmq <20> (integer)
Minimum quality required for a base to pass the NQS filter

-nqsaq <15> (integer)
Minimum quality required for the neighborhood bases to pass NQS

-nqssize <5> (integer)
Size of the neighborhood on each side of the central base that is considered for the NQS filter

-nqvalue <1> (integer)
Q-Score given to a N that gets added by the script in the quality file. Default is 1. The important part here is that all bases that have the nqvalue will get ignored when looking at the neighborhood of the base in the NQS filter. This is to prevent an inserted N from 'flagging down' all adjacent bases. The score 1 is chosen by default because usually no base gets assigned a score this low.

-nosuffix <NULL> (boolean)
By default, in paired reads data the read name will have a suffix added of /1 or /2 depending on the .sam file. This option removes this addition.

-fakequals <0> (integer)
If set to a positive integer number, will fake the quality of all nucleotides to the given value. This speeds up the creation of the .qlx file greatly if you do not care about the quality scores.

8) configPaths:

configPaths.pl is more of an installer than anything else. It will modify the hard-coded paths in all the scripts (vfat.pl, orientContig.pl, contigMerger.pl, fixFrameshifts.pl, annotate.pl, QA_stats.pl, runMosaik.pl) to find the required program on your system.

Command Line :

```
perl configPaths.pl <configfile.txt>
```

Input File:

The input file contains the list of paths for each variable.

scriptpath = '<scriptpath>'
mosaikpath = '<mosaikpath>'
mosaiknetworkpath = '<mosaiknetworkpath>'
musclepath = '<musclepath>'
perlpath = '<perlpath>'
genewisepath = '<genewisepath>'
genewisecfgpath = '<genewisecfgpath>'
samtoolspath = '<samtoolspath>'
Rpath = '<Rpath>'
refDataPath = '<refDataPath>'

All the scripts in this package should be located in **scriptpath**.

mosaikpath is the path for the Mosaik binaries MosaikBuild and MosaikAligner for alignments from contigMerger.pl and vfat.pl. This should be the bin folder of the Mosaik distribution (not necessary if no reads are supplied).

mosaiknetworkpath is the path for Mosaik's network files for alignments from contigMerger.pl and vfat.pl. This should be the networkFile folder of your Mosaik distribution (not necessary if no reads are supplied).

musclepath is the path where you have muscle installed. If you are not sure that your version of muscle will be compatible with the scripts (they were developed on v3.8) you can download v3.8 from their website at http://www.drive5.com/muscle/downloads.htm

perlpath is the path where perl is located

Rpath is the path for R 2.9 or higher to create the coverage and contig mapping plots.

genewisepath is the path for the binaries of your installation of wise2.2. It should be in your wise2.2.0 folder under /src/bin/.

genewisecfgpath is the path for config folder that is required by wise2.2. It should be in your wise2.2.0 folder under /wisecfg/.

samtoolspath is the path for the installation of Samtools for bam/sam file functions. It is required for runMosaik2.pl to run. Not necessary if the scripts are used without reads.

refDataPath is an optional path to locate your reference data. See V-FAT section for details.

Note that if you have the paths in your environment variables and that you can run the different programs like R or muscle without specifying any path when typing a commandline, you do not need to specify a path here either. The only one that absolutely requires a path is scriptpath.

Output File:

There is no output file for configPaths. It will modify the paths in the various scripts in this package.

***** Note : *At the Broad*, the easiest way to setup is this:**

a) set scripts path to the location you want them
b) set muscle path to `"/seq/annotation/bio_tools/muscle/3.8/"`
c) set perl path to `/usr/bin/env`
d) set genewisepath to `/seq/annotation/bio_tools/GeneWise/wise2.2.0/src/bin/`
e) set refDataPath to where the reference data is located (there is an existing folder at : `/seq/viral/analysis/annotation_refSequences/`)
f) BEFORE running the scripts, do:
g) `use .mosaik-1.1.0013`
h) `use R-2.9`
i) `setenv WISECONFIGDIR /seq/annotation/bio_tools/GeneWise/wise2.2.0/wisecfg/`

• File Formats •

a) Genelist and amplicon input files:

The format for these text files is tab-tabulated. It contains the feature name, start and stop position.

Example of a genelist:

```
Gag       12 1514
Pol       1307    4318
Vif       4263    4841
Vpr       4781    5071
Tat_exon1 5052    5266
Tat_exon2 7585    7675
Rev_exon1 5191    5266
Rev_exon2 7585    7859
Vpu       5283    5528
Env       5446    8001
Nef       8003    8623
```

For the genes that have multiple exons, each exon is set on its own line, followed by "_exon#", i.e. [Gene]_exon#. This is the correct way to specify genes with multiple exons. If they are not written this way, they will be treated as separate genes instead.

Example of amplicons input (amplicon names can be anything):

```
Amp1  779   2888
Amp2  2019  4800
Amp3  3678  7134
Amp4  5560  8742
```

b) Peptides folder:

The peptides folder must contain a peptide file in fasta format for each gene, named **[Gene]_pep.fa**, as well as one extra file named **[Virus]_Peptides_Features.txt** containing information on each gene about their product name (for NCBI submission) as well as the required presence of a start/stop codon for this gene. The exact format for this file will be shown in the example below.

The script "translateDna.pl" included in the package can let you generate this peptide folder using the reference and the gene list (see format above). You can use the following command line:

perl translateDna.pl <reference.fasta> <output> -genelist <reference_genelist.txt>

All files names in the example are written in the standard way for the -virus command line option to work.

First is an example of the files found in the folder **HIV_Peptides** :

```
Env_pep.fa
Gag_pep.fa
HIV_Peptides_Features.txt
Nef_pep.fa
Pol_pep.fa
Rev_pep.fa
Tat_pep.fa
Vif_pep.fa
Vpr_pep.fa
Vpu_pep.fa
```

The file `Env_pep.fa` would look like this:

```
>Env
MRVKGIRKNYQHLWRWGTM…
```

The file `HIV_Peptides_Features.txt` would look like this:

```
CDSStatus   Multi
Gene  Start Stop  Product
Gag   1     1     Pr55(Gag)
Pol   0     1     Pol polyprotein
Vif   1     1     Vif
Vpr   1     1     Vpr protein
Env   1     1     Envelope surface glycoprotein gp160, precursor
Tat   1     1     Tat
Rev   1     1     Rev
Vpu   1     1     Vpu
Nef   1     1     27K protein
```

The first line specifies if the virus contains a single CDS (such as DENV or WNV) or multiple CDS (such as HIV). The format is either:

```
CDSStatus   Multi
```
or
```
CDSStatus   Single
```

In this case, a start and stop codon are expected for each gene, except in the case of Pol that does not require a start codon. For a virus like Dengue where you have a single CDS, you would only have 1 in the start codon for the first gene and 1 in the stop codon for the last gene, will all other numbers set at 0.

c) Reads fasta and quals

Reads must be in a fasta/quals format. Read names should not contain spaces or special characters that would cause problems in a hash name as they are used as such in some scripts (_ and – are fine). In general, the following one-liner perl command lines will clean read names and only keep the part before the first space (<reads.fa> and <reads.qual> are your file names):

```
perl -p -i -e 's/>(.+?) .+/>$1/g <reads.fa>
perl -p -i -e 's/>(.+?) .+/>$1/g <reads.qual>
```

The Reads.fa and Reads.qual files should look like this:

Reads.fa:
```
>ReadId1
ATGC…

>ReadId2
TGCA…
...
```

Reads.qual:
```
>ReadId1
20 20 15 25…

>ReadId2
30 30 30 29…
…
```

fastq2fasta.pl and fqpair2fasta.pl:

These scripts included in the package, fastqToFasta.pl and fqpair2fasta.pl, allow to do the conversion from fastq format to reads fasta/qual format. fastqToFasta.pl will take a single fastq file as input, while fqpair2fasta will take 2 paired fastq files (which can be the output of an Illumina run with paired reads for instance).

<u>Command Line :</u>

perl fastq2fasta.pl <reads.fastq> <reads>
perl fqpair2fasta.pl <reads1.fastq> <reads2.fastq> <reads>

In both cases, this will create the files <reads>.fa and <reads>.qual

d) The .qlx file format:

The .qlx file format is a read alignment format that includes quality information for each base as well as a quality flag to indicate if a base passed the Neighborhood Quality Score (NQS) criteria or not, which is used by V-Phaser to call trusted variants. The NQS is calculated for each base of a read depending on its quality and the quality of the bases adjacent to it. The base must pass a minimum quality score of *q* and its *n* adjacent bases on each side must pass a quality score of *q'*. By default those values are *q* = 20, *n* = 5 and *q'* = 15.

The format is the following:

```
>Read [ReadID] [read start] [read stop] [read length] [strand]
[assembly name] [assembly start] [assembly stop]
Read Sequence
Assembly Sequence
NQS String
Quality String (in ASCII format, ASCII character = Quality Score + 33).
```

Note that assembly here is whatever you aligned against.

2 scripts included in the package, samToQlx.pl and qlxToSam.pl, allow one to convert the .qlx files to or from the .sam format (using samtools can then allow conversion to/from bam).

samToQlx.pl and qlxToSam.pl:

These scripts included in the package, samToQlx.pl and qlxToSam.pl, allow one to convert the .qlx files to or from the .sam format (using samtools can then allow conversion to/from bam). RC454 will generate a .qlx read alignment directly. See the section about runMosaik2 for more details about samToQlx. For qlxToSam.pl, the command line is:

Command Line :

```
perl qlxToSam.pl <qlxinput.qlx> <assembly.fasta> <samoutput.sam>
```

translateDna.pl

This script is used to translate a DNA string into its peptide. If you supply a genelist, it will generate a peptides folder with one file for the peptide of each gene.

Command Line :

```
perl translateDna.pl <reference.fasta> <output>
```

Parameters:

-start <0> (integer)
If set, will start the translation of the DNA sequence at the specified base.

-stop <0> (integer)
If set, will stop the translation of the DNA sequence at the specified base.
-genelist <NULL> (string)
If a genelist file is specified with this parameter, the script will create a folder named <output> and put a separate file named [Gene]_pep.fa for the peptide sequence of each gene in the genelist. This can be used to generate the peptides folder for annotate.pl

• Example Data •

The folder TestData included in the vFatPackage contains all files required to test the scripts in this package. The following command line should run the script:

perl vfat.pl -contigs VTest_hiv_contigs.fa –readfa VTest_20k_reads.fa –readq VTest_20k_reads.qual -o VTest_output -virus HIV

The current version contains 2 known minor issues that have not been resolved yet.

1) It is possible for the annotation tool to return a gene as partial when it is full if there is large indels occurring near the ends of the genes when comparing to the reference genome. Such large indels throw off the numbering in genewise. In general, the start and stop position of the gene will be correct, the only problem will be that it will be called partial when it's not. If a gene comes out as partial and you have doubts that it is, check this possibility first.

2) Frameshifts happening in long homopolymers (6+ nucleotides) can occasionally fail to find the correct read window to automatically correct it even if it exists. This is due to alignment issues, so if you have remaining unfixed long homopolymer frameshifts, they might need to be modified manually.

Please report any other issue that you find on our Broad Viral Tool Users forum at:
https://groups.google.com/forum/?fromgroups#!forum/viral-tool-users

Alternatively, you can report them through email at patrickc@broadinstitute.org or mczody@broadinstitute.org.