VirGA Documentation

Release 1.0

Lance Parsons, Yolanda Tafuri, Jacob Shreve, Christopher Bowen

Aug 30, 2017

Contents

VirGA is a collection of scripts and dependencies that work together to autonomously construct semi *de novo* large DNA virus genomes. It can begin with raw next generation sequencing (NGS) reads and concludes with a fully annotated and evaluated genome assembly. Please thoroughly read VirGA's documentation and contact us with any questions you may have: moriah@psu.edu

Contents:

CHAPTER 1

Introduction

High-throughput sequencing has rapidly accelerated the discovery of small RNA and DNA viruses, but has produced markedly fewer large virus draft genomes (>101 kb) due to the limitations of de novo assembly techniques. Despite their relatively small genome sizes, viruses such as herpesviruses, adenoviruses, and large bacteriophages are infrequently assembled into complete genomes. Herpesviruses in particular contain a number of challenging features for de novo assembly, including high G+C content, numerous homopolymers, inverted structural repeats, and many tandem short sequence repeats (SSRs). To address this, we developed a fully automated computational pipeline for Virus Genome Assembly (VirGA). VirGA is comprised of four contiguous steps that conduct 1) sequencing reads preprocessing, 2) de novo assembly, 3) genome linearization and annotation, and 4) assembly assessment to allow high-throughput generation of draft genomes. The pipeline was designed with both desktop PCs and scientific computing clusters in mind, with frequent built-in use of multi-threading, parallelization, and native support for job scheduling and software module systems. Similarly, all steps are optimized for both assembly novices and bioinformaticians alike, with push-button ease of use that still allows for in-depth parameter control when desired. Since VirGA is meant as a rapid solution for generating numerous draft genomes, quality control and reporting strategies are implemented at every step. All scripts and settings used for each run are permanently stored with the output, to allow easy record-keeping and preservation of parameters for future replication. Post-assembly computational remedies ameliorate genome gaps and miss-assemblies, and reference strain comparison identifies gross errors in coding regions. Upon the pipeline's conclusion, a comprehensive HTML report is generated which details assembly metrics and provides helpful visualizations. VirGA's high-throughput and accurate nature will allow traditional virology wet labs to easily generate their own draft genomes and discover phenotypic causation through comparative genomics.

CHAPTER 2

Installation

Installation Methods

1. **Linux**

   General installation instructions for linux based systems, using Ubuntu 12.04 as an example.

2. **Virtual Machine (VM)**

   A large file (~4GB) that can be booted with the free software VirtualBox from Windows/Mac/Linux. Download the virtual machine image here: VirGA_VM_v1-0

Linux Installation

This guide assumes the user is installing the VirGA pipeline on a freshly installed Ubuntu 12.04 box with administrative access. For cluster installation without administrative access, please contact your institutions tech support to coordinate the installation of necessary dependencies.

Dependencies

Being an all-encompasing pipeline, VirGA has numerous bioinformatics software requirements, and many of these packages have their own requirements. The following shoud be installed in order:

1. Java Runtime Environment 1.7.0

1. `sudo apt-get install default-jre`

2. g++ compiler 4.4.7

1. `sudo apt-get install g++`

3. perl-doc 5.10.1

1. `sudo apt-get install perl-doc`

4. biopython 1.64
   1. `sudo apt-get install python-biopython`
5. **python-dev**
   1. `sudo apt-get install python-dev`
6. **python pip**
   1. `sudo apt-get install python-pip`
7. cython 0.21
   1. `sudo pip install cython`
8. paired_sequence_utils
   1. `sudo pip install paired_sequence_utils`
9. **bx-python 0.7.1**
   1. Download bx-python
   2. `tar -zxvf package_name`
   3. `sudo python setup.py install`
10. **pybedtools 0.6**
    1. Download pybedtools
    2. `tar -zxvf package_name`
    3. `sudo python setup.py install`
11. bedtools v2.21.0
    1. `sudo apt-get install bedtools`
12. **Mugsy 2.3**
    1. Download Mugsy
    2. Add the installation folder to `$PATH`
13. **VAMP 0.9.0**
    1. Download VAMP
    2. `tar -zxvf package_name`
    3. `sudo python setup.py install`
14. **Fastx-Toolkit 0.0.13**
    1. Download Fastx-Toolkit
    2. Add the installation folder to `$PATH`
15. **FastQC 0.10.1**
    1. Download FastQC
    2. `chmod 755 fastqc`
    3. Add the installation folder to `$PATH`
16. **R 3.1.1**
    1. Add the following line to the `/etc/apt/sources.list` file:

- `deb http://<my.favorite.cran.mirror>/bin/linux/ubuntu trusty/`

2. `sudo apt-get update`
3. `sudo apt-get install r-base-dev`

17. **Bowtie2 2.2.2**

1. Download Bowtie2
2. Add the installation folder to `$PATH`

18. **SSAKE 3.8**

1. Download SSAKE
2. Add the installation folder to `$PATH`

19. **Samtools 1.1**

1. Download the dependency zlib library
   - `sudo apt-get install zlibc zlib1g zlib1g-dev`
2. Download Samtools
   - `sudo make`
   - Add the installation folder to `$PATH`
   - Add the subdirectory, bcftools, to `$PATH` as well

20. **Freebayes 0.9.14**

1. Install the dependency cmake: `sudo apt-get install cmake`
2. Download Freebayes: `git clone --recursive git://github.com/ekg/freebayes.git`
   - `sudo make`
   - `sudo make install`

21. **Celera 8.1**

1. Download Celera
2. Extract and add the installation folder to `$PATH`

22. **Clustalw2 2.1**

1. Download clustalw2
2. Extract and add the installation folder to `$PATH`

23. **GapFiller 1.10**

1. Obtain a free license and download GapFiller
2. `dos2unix gapfiller.pl`
3. Add the installation folder to `$PATH`

24. **VirGA 1.0**

1. Download VirGA
2. `unzip package_name`
3. Add the VirGA/Pipeline folder to `$PATH`

Verifying the installation

VirGA comes with a script that will check for the ability to carry out the specific tasks requested by the user. This means it may not be necessary to have all dependencies installed, such as GapFiller, if the user doesn't plan on using it. To test for proper installation, follow these steps:

- Execute the following command within the desired working directory, as the VirGA pipeline will create a new file structure repleat with scripts and support files within the current working directory.
- Initiate the pipeline: `VirGA_Pipeline_Build.sh`
- Enter the directory that is created, `VirGA_Pipeline_Directory`, and edit the `VirGA_parameters.ini` file to indicate which steps and substeps are desired, and whether or not to use the PBS/Torque scheduler and the module software system.
- Execute a script that verifies all relavent dependencies: `bash x_scripts/check_for_dependencies.sh`

Virtual Machine (VM) Installation

A barebones installation of VirGA within linux using the Ubuntu 14.04 distribution is available through a VM. The VM itself is a rather large file, approximately 4GB, so please downlaod with a secure and quick connection. Although nearly all of the software dependencies are pre-installed within the VM, licensing issues prevent certain packages from being included. Please follow these instructions for successfully downloading, installing, and running the VM:

1. Download the software VirtualBox, which is free and available for Windows/Mac/Linux.
2. Download the virtual machine image here: VirGA_VM_v1-0
3. Open VirtualBox software and load the VirGA-VM_v1-0 image
4. Configure the options within VirtualBox to designate how many processing cores and how much memory (RAM) to provide to the VM:
   - The more computational resources suppled to the VirGA VM will increase the pipeline's speed but decrease the host computer's speed when the VM is active
5. Activate the VM
6. Use the following username and password to login:
   - Username: `virga-vm`
   - Password: `HSVSequencing2014`
7. Change the password for security's sake
8. Install the dependency **GapFiller**

1. Obtain a free license and download GapFiller into the /Download directory
2. In a terminal window, navigate to the `/Desktop` and run the command: `bash Run_After_Downloading_GapFiller.sh`

9. Confirm successful VirGA installation
   - In a terminal window, navigate to the `/Desktop/Dependencies/dependencies_check` directory
   - Edit the `VirGA_parameters.ini` file to indicate which steps and substeps are desired, and whether or not to use the PBS/Torque scheduler and the module software system
   - Execute a script that verifies all relavent dependencies: `check_for_dependencies.sh`

CHAPTER 3

Usage

Running VirGA

We are now going to run the VirGA pipeline. The first thing we should do is move into a folder that we are ok with VirGA using. When we execute VirGA it will create it's own file system. Consequently it is generally good practice to create a new folder in which to run VirGA. However this is of course not a required step, it is only recommended. The pipeline can be run in any directory.

Step 1:

- `Virga_Pipeline_Build.sh`
- We just executed the build script. This creates our file system.

Step 2:

- `cd VirGA_Pipeline_Directory`
- We just moved into the VirGA Pipeline Directory. We will use the folders and files in here

Step 3:

- `cd x_contaminants`
- `cp /file_path_way_of_contaminants_you_want_to_control_for .`
- Moving the genome files (.fa or .fa.2.bt2) of potential contaminates into the x_contaminants directory

Step 4:

- `cd ..`
- `cd x_inputs`
- `cp /file_path_way_of_output_from_sequencer_that_you_want_to_create_a_genome_from .`
- Moving the output files (.fastq) from the sequencer in to the x_input directory

Step 5:

- `cd ..`
- `cd x_references .`
- `cp /file_path_way_of_reference_sequence_you_want_to_use .`
- Moving the reference sequence's genome file and gene feature format file (.fa, .gff) into the x_references directory.

Step 6:

- `cd ..`
- `vi VirGA_parameters.ini`
- Review the usage parameters in the file and change if needed

Step 7:

- `./VirGA_pipeline`
- Executes the VirGA pipeline, to use qstat -a -u username to make sure a job is submitted you should see the job in your queue

Overview of VirGA Output

After a successful run of the VirGA pipeline, there are a number of output's created by VirGA. Before accessing these outputs first check that all of the job's VirGA submitted to the cluster are completed. To do this issue the qstat -a -u username command and see that the command either returns no jobs in queue or at least no jobs in queue that were created by VirGA. Then to access the various outputs complete the following step.

Command

- `cd file_path_way_of_VirGA_Pipeline_Directory`
- We just moved into the VirGA_Pipeline_Directory

Now within the VirGA_pipeline directory, which we just moved into in the step above, there are a number of places we can go for different outputs. More specifically there are five key places we can go to for output. These are:

- STEP_1–Preprocessing
- STEP_2–Multi_SSAKE_Assembly
- STEP_3–Linearize_and_Annotate
- STEP_4–Assembly_Assessment
- VirGA_Report...
- x_job_files

Each of these directories and zip file has different outputs and a different organization. See the associated pages for details on the specific output files in each of these places.

STEP_1–Preprocessing Output

In order to access the STEP_1–Preprocessing output, we must move into the STEP_1–Preprocessing directory. Assuming you are in the VirGA_Pipeline_Directory complete the following step.

Command

- `cd STEP_1--Preprocessing`
- We just moved into the the STEP_1–Preprocessing directory. We moved into this directory from the VirGA_Pipeline_Directory

Now with in this directory there are a number of additional directories:

- after_metrics
- artifact_filtered_reads
- before_metrics
- clipped_reads
- contaminant_filtered_reads
- properly_paired_reads
- trimmed_reads

See below for more detailed information on the output in each of these directories.

after_metrics

The after_metrics directory has a number of files and directories with in it:

reads_1/2.fastq_after This directory contains the .fastq file for the actual reads that have passed through all of preprocessing. In this directory there are actually two files these files are identical one is just zipped and the other is not.

reads_1/2.fastq_boxplot.png A .png image of a box plot which shows the quality score at each position on a read which is representative of all of the reads.

reads_1/2.fastq.stats A report on various statistics on the reads after preprocessing has been completed.

artifact_filtered_reads

The artifact_filtered_reads directory has two files with in it. These are:

artifact_trim_clip_reads1.fastq

artifact_trim_clip_reads2.fastq

Both these files are fastq files with reads that were just clipped for adapter, trimmed for quality and have now been filtered for artifacts. More specifically each read has been filtered for sequencing artifacts, which means that the sequencing artifacts were removed from each read.

before_metrics

The before metrics folder contains many directories and files. Each of these are the same as the files and directories in the after_metrics directory. The only difference is the reads and information used to generate the files and results here are all generated from reads which have not gone through preprocessing.

clipped_reads

Within the clipped reads directory there are two files. These are:

clip_reads_1.fastq

clip_reads_2.fastq

Both of these .fastq files are the reads from the sequencer after there adapters have been clipped or removed. Whether or not adapters are removed and the sequence of the adapter removed is set on lines 57 - 61 in the VirGA_parameters.ini file that is configured prior to running of the pipe line.

contaminant_filtered_reads

With in the contaminant_filtered_reads directory there are a number of files and directories:

decon_artifact_trim_clip_1/2.fastq This file contains the reads that have been clipped for adapters, trimmed for quality and have now have been filtered for contaminants. With reads identified as contaminants removed from the population of reads.

decon_artifact_trim_clip_1/2.fastq.sam This is the .sam file for the contaminants. Contaminants are identified by mapping all reads to the genome of potential contaminants. Reads that successfully map to the contaminant genomes are identified as contaminates. The .sam files are the associated files with these mapping processes.

properly_paired_reads

With in the trimmed reads directory there are a number of files and directories:

paired_decon_artifact_trim_clip_reads1/2.fastq These are the reads which have now been clipped for adapters, trimmed for quality, filtered for artifacts, decontaminated of any contaminated reads and have now been filtered for lack of a pair. Each read has an associated pair with it from the sequencer. During the prior steps of preprocessing it is possible for one read of a pair to be removed but not the other. This step removes any read which do not have a pair.

Singletons Within the Singletons directory is a single .fastq file. Singletons.fastq. This file is a .fastq file composed of all the reads which did not have another read paired with it after clipping trimming and filtering for contaminates.

trimmed_reads

With in the trimmed reads directory there are two files:

trim_clip_reads_1.fastq

trim_clip_reads_2.fastq

Both of these .fastq files are the reads that were just clipped for adapters and are now also trimmed. In the process of trimming the segment at the end of each read where the quality scores of bases begin to drop is removed.

STEP_2–Multi_SSAKE_Assembly Output

In order to access the STEP_2–Multi_SSAKE_Assembly output, we must move into the STEP_2–Multi_SSAKE_Assembly directory. Assuming you are in the VirGA_Pipeline_Directory complete the following step.

Command:

- `cd STEP_2--Multi_SSAKE_Assembly`
- We just moved into the the STEP_2–Multi_SSAKE_Assembly directory

Now with in this directory there are a number of additional directories and files:

- Celera
- clean_contigs.fa
- Multi_SSAKE
- rough_contigs.fa

See below for more detailed information on the output in each of these directories and files.

Celera

Celera takes the many small contigs and creates fewer larger contigs from them. In the Celera directory there are the supporting files from Celera.

clean_contigs.fa The clean contigs.fa file is a fasta file which contains the fewer and larger contigs created by Celera from the rough contigs SSAKE created.

Multi_SSAKE

With in the Multi-SSAKE directory there are a number of directories and files:

16-0 / 16-4 / 19-0 / 20-0 / 20-4 / 29-0 / 29-4 (kmer/trimmings pairs may differ) With in each of these directories is the associated information for each of these runs of SSAKE. The VirGA pipeline uses SSAKE to create it's contigs from the preprocessed reads. It also does not use just one run of SSAKE. VirGA runs SSAKE 8 times each time with a different camer length (first number) and a different trim length (second number). It then takes all of the contigs created by each of these runs of SSAKE, passes them thought Celera and uses them all in Step 3 where they are alined to the reference genome and ultimately linearized. With that the camer length and trim length of each of these 8 runs of SSAKE are adjustable at line 96 of the VirGA_parameter.ini file which is configured prior to running VirGA.

paired_decon_artifact_trim_clip..1/2.fastq These are the same .fastq files created at the end of preprocessing. These .fastq file just have small formatting differences in them which are required and specific to SSAKE.

Singletons.fasta These are the same files created at the end of preprocessing. These .fasta and .fastq files are just have small formatting difference in them which are required and specific to SSAKE.

rough_contigs.fa The rough contigs.fa file is a fasta file which contains the many small contigs produced by the SSAKE.

STEP_3–Linerarize_and_Annotate Output

In order to access the STEP_3–Linerarize_and_Annotate output, we must move into the STEP_3–Linerarize_and_Annotate directory

Command:

- `cd STEP_3--Linerarize_and_Annotate`
- We just moved into the the STEP_3–Linerarize_and_Annotate directory

Now with in this directory there are a number of additional directories and files:

- assembled_genome_before_gapfiller.fa
- assembled_genome_before_gapfiller.gff
- assembled_genome.fa
- assembled_genome.gff
- Compare_genomes
- GapFiller
- MUGSY_and_Maf-Net

See below for more detailed information on the output in each of these directories and files.

assembled_genome_before_gapfiller.fa/.gff The assembled genome sequence and general feature format file for the assembled genome before gap filler is the fasta and gff file for the linearized contig's prior to the running of gap filler.

assembled_genome.fa/.gff The assembled genome sequence and general feature format file for the assembled genome is the fasta and gff file for the linearized contig's which have have also had as the gaps between the configs filled by gap filler.

Compare_genome

With in the Comapare_genome directory, there are many .fasta file and additional files. The majority of these files are files related to the running of the Compare genome program, which uses the reference genome to create the annotation (general feature format file) for the new genome created by VirGA.

GapFiller

Within the GapFiller directory there is all of the supporting output files for Gap Filler. The purpose of GapFiller is to fill the spaces between the linearized blocks that Maf-Net creates using the reference genome.

MUGSY_and_Maf-Net

With in the MUSY_and_Mad_Net directory there is all of the supporting output and files for MUSGY and for Maf-Net. The files of interest generated by these steps are described above and are not located in the exact directory. With that this use of space is more likely best used to describing in concept what MUSGY and Maf-Net do.

MUGSY is an alignment software which creates synteny blocks from the contigs and the reference genome. These synteny blocks are then used by Maf-Net which will pick the best synteny blocks with respect to overlap and will further then linerarize the blocks, which can then be passed through GapFiller which will use the reference sequence to try to fill gaps between the blocks.

STEP_4–Assembly_Assessment Output

In order to access the STEP_4–Assembly_Assessment output, we must move into the STEP_4–Assembly_Assessment directory.

Command:

- `cd STEP_4--Assembly_Assessment`

- We just moved into the the STEP_4–Assembly_Assessment directory

Now with in this directory there are a number of additional directories and files:

- Bowtie2
- coverage_graphic.png
- false_variants_that_were_corrected.vcf
- feature_report.txt
- Freebayes_SNPs.vcf
- full-lenght_genome.fa
- full-lenght_genome.gff
- genome.fa
- genome.fa.fai
- genome.gff
- low_coverage.gff
- no_converage.gff
- Samtools_SNPs.vcf

See below for more detailed information on the output in each of these directories and files.

coverage_graphic.png A png image of the coverage plot for the assembled genome. Specifically it is the depth of coverage at each position on the genome. Depth of coverage is plotted on a log scale and depth of coverage is represented in number of reads. Additionally on the coverage .png there are a few tracks above and below the main depth of coverage plot, which indicate some additional features of interest.

false_variants_that_were_corrected.vcf A variant call format file which keeps a record of all of the false variants in the assembled genome that were corrected for.

feature_report.txt A report on each of the genes in the genome of the assembled stain.

Freebayes_SNPs.vcf A variant call format file which keep a record of the variants found by Freebayes in the assembled genome.

full-lenght_genome.fa/.gff The full length fasta file (.fa) and general feature format file(.gff) are the full length genome and annotation including the regions of invert repeat. These files are specific to the Herpes Simplex Virus, as it has areas of inverted repeats.

genome.fa/.fa.fai/.gff The fasta file (.fa), indexed fasta file (.fa.fai) and general feature format file (.gff) are the final assembled genome and annotation file created from the initial input of reads from a sequencer. Additionally the indexed fasta file is included here for convenient when attempting to visualize the genome from a genome browser.

low_coverage.gff A general feature format file for the areas of the genome which have low coverage. What is considered low coverage is set by you when you edit the VirGA_parameters.ini file. The threshold for what is low coverage is set on line 167 of the VirGA_parameters.ini file

no_coverage.gff A general feature format file for the areas of the genome which have none or zero coverage

Samtools_SNPs.vcf A variant call format file which keep a record of the variants found by Samtools in the assembled genome.

Bowtie2

The Bowtie 2 directory has many files in it. Many of them are specific files to Bowtie 2. Of the files the most notable files in this directory are the corrected.pileup file. This file is the pile up file which has the coverage at each position in the genome. If is the corrected pile up file because the file has been corrected so that position where there is no coverage have a zero in the coverage field instead of simply begin excluded from the file. There is also the .bam file located in this folder which is useful for loading the assembled genome into a genome browser.

VirGA_Report Output

In order to access the VirGA_Report.zip output, it is easiest if we first pull a copy of the zip folder to our local machine. Where we can then unzip the folder and look into the content of the unzipped folder. To do this follow the steps below:

Step 1:

- `scp file_path_way_of_VirGA_Report.zip file_path_way_where_you_want_the_zip_file_to_go`
- We just copied the VirGA_Repor.zip file to the designated location above. If this is unclear visit the transferring files to and from a cluster page for general information on moving files to and from the cluster

Step 2:

- `unzip VirGA_Report.zip`
- We just unzipped the VirGA_Report.zip file

Step 3:

- `cd VirGA_Report`
- We just moved into the directory created when we unzipped the zip file

Now with in this directory there are a number of additional directories and files:

- Aligments
- Assembled_genome
- QC
- Reference
- Variants
- VirGA_parameters.txt
- VirGA_Report.html

See below for more detailed information on the output in each of these directories and files.

VirGA_Report.html The VirGA_report.html is an html file that when opened in a web browser will provide a nice graphical user interface that will allow for access to all of the outputs created by the VirGA pipeline.

VirGA_parameters.txt The VirGA_parameters.txt text file is a copy of the VirGA_parameters.ini file that you edited during the set up before submission of the pipeline to the cluster. This file is useful for referencing the specific parameters you chose for the pipeline to run within.

Alignments

The Alignments directory has a number of directories with in it:

AA The AA directory contains the CLUSTAL alignments between the reference stain placed in the x_reference directory when setting up the pipeline and the assembled genome created by VirGA. The amino acid alignments are organized by protein. With each alignment file (.aln) corresponding to a protein. The CLUSTAL amino acid alignment is a quality control check. Comparing the reference sequence and the assembled genome. It also servo to check for gross variations or differences between the assembled genome and the reference genome, with respect to amino acid composition.

Gene The Gene directory contains the CLUSTAL aliments between the Genes of the reference stain placed in the x_reference directory when setting up the pipeline and the assembled genome created by VirGA. The gene alignments are organized by gene. With each alignment file (.aln) corresponding to a gene. The CLUSTAL gene alignment is a quality control check. Comparing the reference and the assembled genome. It also serves to check for gross variations or differences between the assembled genome and the reference genome, with respect to gene nucleotide composition.

ORF The ORF directory contains the CLUSTAL aliments between the open reading frames of the reference stain placed in the x_reference directory when setting up the pipeline and the assembled genome created by VirGA. The ORF aliments are organized by open reading frames. With each aligment file (.aln) corresponding to an open reading frame. The CLUSTAL open reading frame alignment is a quality control check. Comparing the reference and the assembled genome. It also serves to check for gross variations or differences between the assembled genome and the reference genome, with respect to open reading frames.

Other The Other directory contains the CLUSTAL alignments between the reference stain placed in the x_reference directory when setting up the pipeline and the assembled genome created by VirGA. The Other alignments are organized by there id in the general feature file (.gff) for the genomes. What Other is considered, is anything within the general feature format file (.gff) which is not included in the amino acid alignment, gene alignment or open reading frame alignment, With this each alignment file (.aln) corresponds to labeled identify of the feature in the general feature file (.gff). The CLUSTAL other alignment is a quality control check. Comparing the reference sequence and the assembled genome. It also serves to check for gross variations or difference between the assembled genome and the reference genome with respect to the other features aligned.

Assembled_genome

The Assembled_genome directory has a number of files with in it. More detailed information on each of these files:

coverage_graphic.png The coverage_graphic.png is a png image of the coverage plot for the assembled genome. Specifically it is the depth of coverage at each position on the genome. Depth of coverage is plotted on a log scale and depth of coverage is represented in number of reads. Additionally on the coverage .png there are a few tracks above and below the main depth of coverage plot, which indicate some additional features of interest.

full-lenght_genome.fa/.gff The full length fasta file (.fa) and general feature format file(.gff) are the full length genome and annotation including the regions of invert repeat. These files are specific to the Herpes Simplex Virus, as it has areas of inverted repeats.

genome.fa/.fa.fai/.gff The fasta file (.fa), indexed fasta file (.fa.fai) and general feature format file (.gff) are the final assembled genome and annotation file created from the initial input of reads from a sequencer. Additionally the indexed fasta file is included here for convenient when attempting to visualize the genome from a genome browser.

low_coverage.gff The low_coverage.gff is a general feature format file for the areas of the genome which have low coverage. What is considered low coverage is set by you when you edit the VirGA_parameters.ini file. The threshold for what is low coverage is set on line 167 of the VirGA_parameters.ini file

no_coverage.gff The no_coverage.gff is a general feature format file for the areas of the genome which have none or zero coverage

QC

The QC or quality control directory has two directories with in it.

After

With in the After directory there are a number of files and directories with information on the reads (generated directly from the sequencer) after preprocessing has been completed. More specificity the information generated on the reads after preprocessing includes the following directories and files:

- reads_1.fastq_boxplot.png
- reads_2.fastq_boxplot.png
- paired_decon_artifact_trim_clip_reads_1_fastqc
- paired_decon_artifact_trim_clip_reads_2_fastqc

More detailed information on each of these directories and files:

reads_1/2.fastq_boxplot.png These are two sets of box plots. One for each direction polymerase works in when sequencing a particular read. The box plots show the quality score at each position on a 300 base pair representative off all reads collectively. These plots are useful for obtaining a visualization and general idea of the position on the read where quality becomes to low for confidence in the sequenced output.

paired_decon_artifact_trim_clip_reads_1_fastqc This directory contains the .fastq file for the actual reads after that have passed through all of preprocessing. In this directory there are actually two files these files are identical one is just zipped and the other is not.

Before

Wth in the Before directory there are a number of files and directories with information on the reads (generated directly from the sequencer) before preprocessing has occurred. More specificity the information generated on the reads before preprocessing is exactly the same as above, except the reads used to generate the all of the files are the reads before reprocessing.

Reference

With in the reference directory is the reference sequence (.fa) and general feature form format file (.gff) for the reference sequence that you originally placed in the x_reference directory when you were setting up the pipe line.

Variants

The variants directory has a few additional files within

- false_variants_that_were_corrected.vcf
- Freebayes_SNPs.vcf
- Samtools_SNPs.vcf

More detailed information on each of these directories:

false_variants_that_were_corrected.vcf The false_variants_that_were_corrected.vcf is a variant call format file which keeps a record of all of the false variants in the assembled genome that were corrected for.

Freebayes_SNPs.vcf / Samtools_SNPs.vcf The Freebayes_SNPs.vcf / Samtools_SNPs.vcf are variant call format files which keep a record of the variants found by the Freebayes and Samtools in the assembled genome.

CHAPTER 4

Examples

Test case #1: HSV1 assembly using 250,000 paired-end reads

This scenario uses VirGA's entire pipeline with all steps included. The 250,000 reads are a random sub-set of paired-end reads from a pool of ~20M reads produced when sequencing the straing HSV-F-large. A standard VirGA run would utilize all ~20M reads, but the number of significantly reduced for the purpose of this demonstration. The cells used to culture the virus were from the Green Monkey Kidney line, so the Macaque genome is used to filter contaminating sequence. Since HSV-F-large strain is an HSV1 virus, the common HSV1 reference of HSV1-17 that comes prepackaged with VirGA was used as the reference.

The following steps should be performed in order.

1. With an empty working directory, execute the command: `VirGA_Pipeline_Build`
2. `cd` into the newly created directory, `VirGA_Pipeline_Directory`, and browse its contents: `ls -la`:

   VirGA_pipeline VirGA_parameters.ini x_input x_reference x_contaminants
3. Copy the reads into the `x_input` directory:

- `cp /path_to_reads/read_1.fastq x_input`
- `cp /path_to_reads/read_2.fastq x_input`

4. Copy the reference genome and annotation files into the `x_references` directory:

- `cp /path_to_VirGA/References/HSV-17_trimmed.fa x_references`
- `cp /path_to_VirGA/References/HSV-17_trimmed.gff x_references`

5. Copy the BLAST-indexed contaminants database into the `x_contaminants` directory:

- `cp /path_to_contaminants/my_BLAST_indexed_database_files* x_contaminants`

Test case #2: Adding wet-bench corrections to assembled genome

This scenario uses only certain portions of the VirGA pipeline to reannotate the recently assembled genome while included some wet-bench va

CHAPTER 5

Contact, Citing

Contact

We welcome your comments and questions! Bioinformatics support is offered through the Szpara Lab, please contact Daniel Renner at dwr19@psu.edu.

Citing

VirGA is current in prep. Lance Parsons, Yolanda Tafuri, Jacob Shreve, Christopher Bowen, Mackenzie Shipley, L.W. Enquist, Moriah Szpara (2014). Rapid genome assembly and comparison maps mutations linked to phenotypic change in human herpesviruses. (*In prep, M.Bio*)