snppipeline Documentation

Release 0.7.0

Errol Strain, Yan Luo, James Pettengill, Hugh A. Rand, Steve Davi

Jun 06, 2017

Contents

Contents:

CHAPTER 1

CFSAN SNP Pipeline

The CFSAN SNP Pipeline is a Python-based system for the production of SNP matrices from sequence data used in the phylogenetic analysis of pathogenic organisms sequenced from samples of interest to food safety.

The SNP Pipeline was developed by the United States Food and Drug Administration, Center for Food Safety and Applied Nutrition.

- Free software: See license below.
- Documentation: http://snp-pipeline.readthedocs.io/en/latest/readme.html
- Source Code: https://github.com/CFSAN-Biostatistics/snp-pipeline
- PyPI Distribution: https://pypi.python.org/pypi/snp-pipeline

Introduction

The CFSAN SNP Pipeline uses reference-based alignments to create a matrix of SNPs for a given set of samples. The process generally starts off by finding a reference that is appropriate for the samples of interest, and collecting the sample sequence data into an appropriate directory structure. The SNP pipeline can then be used to perform the alignment of the samples to the reference. Once the sample sequences are aligned, a list of SNP positions is generated. The list of SNP positions is then used in combination with alignments of the samples to the reference sequence to call SNPs. The SNP calls are organized into a matrix containing (only) the SNP calls for all of the sequences.

This software was developed with the objective of creating high quality SNP matrices for sequences from closely-related pathogens, e.g., different samples of Salmonella enteriditis from an outbreak investigation. The focus on closely related sequences means that this code is not suited for the analysis of relatively distantly related organisms, where there is not a single reference sequence appropriate for all the organisms for which an analysis is desired.

The CFSAN SNP Pipeline is written in a combination of bash and python. The code (including the bash scripts) is designed to be straighforward to install. Scripts are provided to run the Python code from the command line. A configuration file supports customizing the behavior of the pipeline. In situations where additional customization is desired, the code is not highly complex and should be easy to modify as necessary.

Examples of using the code are provided. These examples serve as both unit tests, and as examples that can be modified to work on other data sets of interest.

Citing SNP Pipeline

Please cite the publication below:

> Davis S, Pettengill JB, Luo Y, Payne J, Shpuntoff A, Rand H, Strain E. (2015) CFSAN SNP Pipeline: an automated method for constructing SNP matrices from next-generation sequence data. PeerJ Computer Science 1:e20 https://doi.org/10.7717/peerj-cs.20

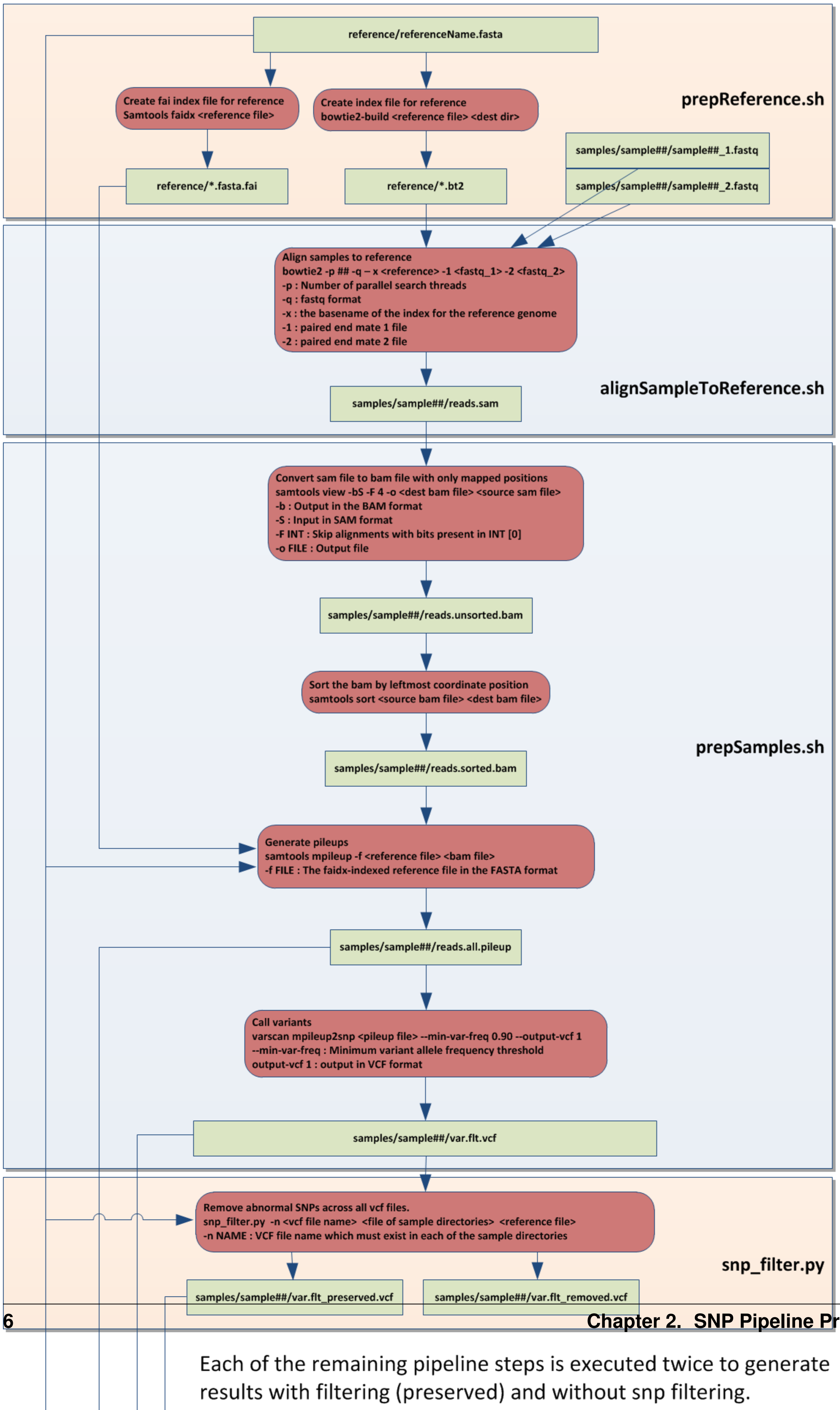
License

See the LICENSE.txt file included in the SNP Pipeline distribution.

CHAPTER 2

SNP Pipeline Processes

The drawing below depicts the processes, files, and data flows within the SNP Pipeline.


reference/referenceName.fasta
Create fai index file for reference
Samtools faidx <reference file>
Create index file for reference
bowtie2-build <reference file> <dest dir>
prepReference.sh
samples/sample##/sample##_1.fastq
samples/sample##/sample##_2.fastq
reference/*.fasta.fai
reference/*.bt2
Align samples to reference
bowtie2 -p ## -q – x <reference> -1 <fastq_1> -2 <fastq_2>
-p : Number of parallel search threads
-q : fastq format
-x : the basename of the index for the reference genome
-1 : paired end mate 1 file
-2 : paired end mate 2 file
samples/sample##/reads.sam
alignSampleToReference.sh
Convert sam file to bam file with only mapped positions
samtools view -bS -F 4 -o <dest bam file> <source sam file>
-b : Output in the BAM format
-S : Input in SAM format
-F INT : Skip alignments with bits present in INT [0]
-o FILE : Output file
samples/sample##/reads.unsorted.bam
Sort the bam by leftmost coordinate position
samtools sort <source bam file> <dest bam file>
samples/sample##/reads.sorted.bam
prepSamples.sh
Generate pileups
samtools mpileup -f <reference file> <bam file>
-f FILE : The faidx-indexed reference file in the FASTA format
samples/sample##/reads.all.pileup
Call variants
varscan mpileup2snp <pileup file> --min-var-freq 0.90 --output-vcf 1
--min-var-freq : Minimum variant allele frequency threshold
output-vcf 1 : output in VCF format
samples/sample##/var.flt.vcf
Remove abnormal SNPs across all vcf files.
snp_filter.py -n <vcf file name> <file of sample directories> <reference file>
-n NAME : VCF file name which must exist in each of the sample directories
snp_filter.py
samples/sample##/var.flt_preserved.vcf
samples/sample##/var.flt_removed.vcf
Each of the remaining pipeline steps is executed twice to generate
results with filtering (preserved) and without snp filtering.

CHAPTER 3

Installation

The SNP Pipeline software package consists of python scripts and shell scripts with dependencies on executable programs launched by the scripts.

Step 1 - Operating System Requirements

The SNP Pipeline runs in a Linux environment. It has been tested on the following platforms:

- Red Hat
- CentOS
- Ubuntu

Step 2 - Executable Software Dependencies

You should have the following software installed before using the SNP Pipeline.

- Bowtie2, a tool for aligning reads to long reference sequences
- SMALT, a tool for aligning reads to long reference sequences
- SAMtools, utilities for manipulating alignments in the SAM format
- VarScan, a tool to detect variants in NGS data
- tabix, a generic indexer for tab-delimited genome position files
- bgzip, part of the tabix package, bgzip is a block compression utility
- BcfTools, utilities for variant calling and manipulating VCFs and BCFs
- fastq-dump, an SRA Toolkit utility for fetching samples from NCBI SRA

Note: you will need either Bowtie2 or SMALT. You do not have to install both. However, the included result files were generated with Bowtie2. Your results may differ when using SMALT.

Step 3 - Environment Variables

Define the CLASSPATH environment variable to specify the location of the VarScan jar file. Add the following (or something similiar) to your .bashrc file:

```
export CLASSPATH=~/software/varscan.v2.3.9/VarScan.v2.3.9.jar:$CLASSPATH
```

Step 4 - Python

The SNP pipeline requires python version 2.6, 2.7, 3.3, 3.4, or 3.5. The pipeline has not been tested on other python versions. If you do not already have python installed, you should install version 2.7. You can either build from source or install a precompiled version with your Linux package manager.

Step 5 - Pip

This can be a troublesome installation step – proceed with caution. The pip tool is used to install python packages including the snp-pipeline and other packages used by the snp-pipeline. Some newer versions of Python include pip. Check to see if pip is already installed:

```
$ pip -V
```

If pip is not already installed, proceed as follows:

```
Download get-pip.py from https://pip.pypa.io/en/latest/installing.html#install-pip
$ python get-pip.py --user
```

Note: avoid using sudo when installing pip. Some users have experienced problems installing and loading packages when pip is installed using sudo.

Step 6 - Python Package Dependencies

For the most part, the installer automatically installs the necessary python packages used by snp-pipeline. However, not all python packages can be reliably installed automatically. The packages listed below may need to be manually installed if automatic installation fails. You can either install these packages now, or hope for the best and manually install later if the automatic installation fails.

- Biopython, a set of tools for biological computation written in Python.

Step 7 - Install the SNP Pipeline Python Package

There is more than one way to install the SNP Pipeline depending on whether you intend to work with the source code or just run it.

Installation Method 1 for Most Users

This is the recommended installation method for new users.

If you want to run the software without viewing or changing the source code, follow the instructions below.

At the command line:

```
$ pip install --user snp-pipeline
```

Update your .bashrc file with the path to user-installed python packages:

```
export PATH=~/.local/bin:$PATH
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv snp-pipeline
$ pip install snp-pipeline
```

Installation Method 2 for Software Developers

If you intend to work with the source code in the role of a software developer, you should clone the GitHub repository as described in the *Contributing* section of this documentation.

Upgrading SNP Pipeline

If you previously installed with pip, you can upgrade to the newest version from the command line:

```
$ pip install --user --upgrade snp-pipeline
```

Uninstalling SNP Pipeline

If you installed with pip, you can uninstall from the command line:

```
$ pip uninstall snp-pipeline
```

Tips

There is a dependency on the python psutil package. Pip will attempt to install the psutil package automatically when installing snp-pipeline. If it fails with an error message about missing Python.h, you will need to manually install the python-dev package. In Ubuntu, use this command:

```
$ sudo apt-get install python-dev
```

CHAPTER 4

Usage

The SNP Pipeline is run from the Unix command line. The pipeline consists of a collection of shell scripts and python scripts.

Script	Description
copy_snppipeline_data.py	Copies supplied example data to a work directory
run_snp_pipeline.sh	This do-it-all script runs all the other scripts listed below, comprising all the pipeline steps
prepReference.sh	Indexes the reference genome
alignSampleToReference.sh	Aligns samples to the reference genome
prepSamples.sh	Finds variants in each sample
snp_filter.py	Remove SNPs in abnormal regions.
create_snp_list.py	Combines the SNP positions across all samples into a single unified SNP list file
create_snp_pileup.py	Deprecated – this command is not used by the pipeline since v0.4.0. Replaced by call_consensus.py Creates the SNP pileup file for a sample – a subset of the pileup file at only the positions where SNPs were called in any of the samples
call_consensus.py	Calls the consensus SNPs for each sample
create_snp_matrix.py	Creates a matrix of SNPs across all samples
calculate_snp_distances.py	Computes the SNP distances between all pairs of samples
create_snp_reference_seq.py	Writes the reference sequence bases at SNP locations to a fasta file

Inputs

Before using the SNP Pipeline, make sure your input data is organized and named the way the pipeline expects. Follow these guidelines:

- No spaces in file names and directory names.
- A fasta genome reference file must exist in a separate directory.
- The samples must be organized with a separate directory for each sample. Each sample directory should contain the fastq files for that sample. The name of the directory should match the name of the sample. When using paired-end fastq files, the forward and reverse files must be in the same directory.
- The script needs to know how to find all the samples. You have two choices:
  1. You can organize all the sample directories under a common parent directory.
  2. You can have sample directories anywhere you like, but you will need to create a file listing the path to all the sample directories.
- The sample fastq files must be named with one of the following file patterns: (*.fastq, *.fq, *.fastq.gz, *.fq.gz). It's okay if different samples are named differently, but the two mate files of paired-end samples must be named with the same extension.
- If there is an outgroup among samples, a file containing the sample ids of the outgroup samples must be created in advance, and the relative or absolute path of this file should be specified in the parameter "RemoveAbnormalSnp_ExtraParams" in the configuration file "snppipeline.conf" (see the description of the parameter "RemoveAbnormalSnp_ExtraParams").

Outputs

By default, the SNP Pipeline generates the following output files. If you need more control over the output, you can run the pipeline one step at a time. See *Step-by-Step Workflows*.

- `snplist.txt` : contains a list of the high-confidence SNP positions identified by the phase 1 SNP caller (VarScan) in at least one of the samples. These are the only positions where the consensus caller subsequently looks for SNPs in all samples. The consensus caller often finds SNPs at the same positions in other samples, and those additional SNPs are not listed in the snplist.txt file. While the snplist.txt file has an accurate list of SNP positions, it does not contain the final list of samples having SNPs at those positions. If you need the final set of SNPs per sample, you should not use the snplist.txt file. Instead, refer to the snpma.fasta file or the snpma.vcf file. The corresponding `snplist_preserved.txt` file is produced when snp filtering removes the abnormal snps.
- `consensus.fasta` : for each sample, the consensus base calls at the high-confidence positions where SNPs were detected in any of the samples. The corresponding `consensus_preserved.fasta` file is produced when snp filtering removes the abnormal snps.
- `consensus.vcf` : for each sample, the VCF file of SNPs called, as well as failed SNPs at the high-confidence positions where SNPs were detected in any of the samples. The corresponding `consensus_preserved.vcf` file is produced when snp filtering removes the abnormal snps.
- `snpma.fasta` : the SNP matrix containing the consensus base for each of the samples at the high-confidence positions where SNPs were identified in any of the samples. The matrix contains one row per sample and one column per SNP position. Non-SNP positions are not included in the matrix. The matrix is formatted as a fasta file, with each sequence (all of identical length) corresponding to the SNPs in the correspondingly named sequence. The corresponding `snpma_preserved.fasta` file is produced when snp filtering removes the abnormal snps.

- `snp_distance_pairwise.tsv` : contains the pairwise SNP distance between all pairs of samples. The file is tab-separated, with a header row and three columns identifing the two sequences and their distance. The corresponding `snp_distance_pairwise_preserved.tsv` file is produced when snp filtering removes the abnormal snps.
- `snp_distance_matrix.tsv` : contains a matrix of the SNP distances between all pairs of samples. The file is tab-separated, with a header row and rows and columns for all samples. The corresponding `snp_distance_matrix_preserved.tsv` file is produced when snp filtering removes the abnormal snps.
- `snpma.vcf` : contains the merged multi-sample VCF file identifying the positions and snps for all samples. The corresponding `snpma_preserved.vcf` file is produced when snp filtering removes the abnormal snps.
- `referenceSNP.fasta` : a fasta file containing the reference sequence bases at all the SNP locations. The corresponding `referenceSNP_preserved.fasta` file is produced when snp filtering removes the abnormal snps.
- `metrics` : for each sample, contains the size of the sample, number of reads, alignment rate, pileup depth, and number of SNPs found.
- `metrics.tsv` : a tab-separated table of metrics for all samples containing the size of the samples, number of reads, alignment rate, pileup depth, and number of SNPs found.
- `error.log` : a summary of errors detected during SNP Pipeline execution

All-In-One SNP Pipeline Script

Most users should be able to run the SNP Pipeline by launching a single script, `run_snp_pipeline.sh`. This script is easy to use and works equally well on your desktop workstation or on a High Performance Computing cluster. You can find examples of using the script in the sections below.

If you need more flexibility, you can run the individual pipeline scripts one step at a time. See *Step-by-Step Workflows*.

Logging

When the SNP Pipeline is launched with the `run_snp_pipeline.sh` script, it generates log files for each processing step of the pipeline. The logs for each pipeline run are stored in a time-stamped directory under the output directory. If the pipeline is re-run on the same samples, the old log files are kept and a new log directory is created for the new run. For example, the output directory might look like this after two runs:

```
drwx------ 2 me group 4096 Oct 17 16:37 logs-20141017.154428/
drwx------ 2 me group 4096 Oct 17 16:38 logs-20141017.163848/
drwx------ 2 me group 4096 Oct 17 16:37 reference/
-rw------- 1 me group  194 Oct 17 16:38 referenceSNP.fasta
-rw------- 1 me group  182 Oct 17 16:38 referenceSNP_preserved.fasta
-rw------- 1 me group  104 Oct 17 16:38 sampleDirectories.txt
drwx------ 6 me group 4096 Oct 17 16:37 samples/
-rw------- 1 me group 7216 Oct 17 16:38 snplist.txt
-rw------- 1 me group 6824 Oct 17 16:38 snplist_preserved.txt
-rw------- 1 me group  708 Oct 17 16:38 snpma.fasta
-rw------- 1 me group  682 Oct 17 16:38 snpma_preserved.fasta
```

A log file is created for each step of the pipeline for each sample. For performamnce reasons, the samples are sorted by size and processed largest first. This sorting is reflected in the naming of the log files. The log files are named with a suffix indicating the sample number:

```
-rw------- 1 me group  1330 Oct 17 16:37 alignSamples.log-1
-rw------- 1 me group  1330 Oct 17 16:37 alignSamples.log-2
-rw------- 1 me group  1330 Oct 17 16:37 alignSamples.log-3
-rw------- 1 me group 12045 Oct 17 16:37 prepReference.log
-rw------- 1 me group  1686 Oct 17 16:37 prepSamples.log-1
-rw------- 1 me group  1686 Oct 17 16:37 prepSamples.log-2
-rw------- 1 me group  1686 Oct 17 16:37 prepSamples.log-3
-rw------- 1 me group   983 Oct 17 16:37 snpList.log
-rw------- 1 me group   983 Oct 17 16:37 snpList_preserved.log
-rw------- 1 me group  1039 Oct 17 16:37 snpMatrix.log
-rw------- 1 me group  1039 Oct 17 16:37 snpMatrix_preserved.log
-rw------- 1 me group   841 Oct 17 16:37 snpPileup.log-1
-rw------- 1 me group   841 Oct 17 16:37 snpPileup.log-2
-rw------- 1 me group   841 Oct 17 16:37 snpPileup.log-3
-rw------- 1 me group   806 Oct 17 16:37 snpReference.log
-rw------- 1 me group   806 Oct 17 16:37 snpReference_preserved.log
```

To determine which samples correspond to which log files, you can either grep the log files for the sample name or inspect the sorted sampleDirectories.txt file to determine the sequential position of the sample. The file names are consistent regardless of whether the pipeline is run on a workstation or HPC cluster.

In addition to the processing log files, the log directory also contains a copy of the configuration file used for each run – capturing the parameters used during the run.

Mirrored Inputs

When the SNP Pipeline is launched with the `run_snp_pipeline.sh` script, it has the optional capability to create a mirrored copy of the input fasta and fastq files. You might use this feature to avoid polluting the reference directory and sample directories with the intermediate files generated by the snp pipeline. The mirroring function can either create normal copies of the files, or it can create links to the original files – saving both time and disk space. With linked files, you can easily run multiple experiments on the same data or different overlapping sets of samples without having duplicate copies of the original sample files. See the *run_snp_pipeline.sh* command reference for the mirroring syntax.

The mirroring function creates a "reference" subdirectory and a "samples" subdirectory under the main output directory. One directory per sample is created under the "samples" directory. The generated intermediate files are placed into the mirrored directories, not in the original locations of the inputs. The SNP Pipeline attempts to preserve the time stamps of the original files in the mirrored directories.

Keep in mind the following limitations when mirroring the inputs.

- Some file systems do not support soft (symbolic) links. If you attempt to create a soft link on a file system without the capability, the operation will fail with an error message.
- Hard links cannot be used to link files across two different file systems. The original file and the link must both reside on the same file system.
- Normal file copies should always work, but the copy operation can be lengthy and the duplicate files will consume extra storage space.

High Performance Computing

The SNP Pipeline can be executed on a High Performance Computing cluster. The Torque and Grid Engine job queue managers are supported.

Torque

To run the SNP Pipeline on torque:

```
run_snp_pipeline.sh -Q torque -s mySamplesDir myReference.fasta
```

You may need to change the `Torque_StripJobArraySuffix` configuration parameter if you see qsub illegal dependency errors.

You can pass extra options to the Torque qsub command by configuring the `Torque_QsubExtraParams` parameter in the configuration file.

Grid Engine

To run the SNP Pipeline on grid engine you must use a configuration file to specify the name of your parallel environment.

Grab the default configuration file:

```
copy_snppipeline_data.py configurationFile
```

Edit the snppipeline.conf file and make the following change:

```
GridEngine_PEname="myPE" # substitute the name of your PE
```

You may also need to change the `GridEngine_StripJobArraySuffix` configuration parameter if you see qsub illegal dependency errors.

Then run the pipeline with the -c and -Q command line options:

```
run_snp_pipeline.sh -c snppipeline.conf -Q grid -s mySamplesDir myReference.fasta
```

You can pass extra options to the Grid Engine qsub command by configuring the `GridEngine_QsubExtraParams` parameter in the configuration file. Among other things, you can control which queue the snp-pipeline will use when executing on an HPC with multiple queues.

See also: *Performance*.

Tool Selection

The SNP Pipeline lets you choose either the Bowtie2 aligner or the Smalt aligner. Your choice of aligner, as well as the command line options for the aligner are specified in the SNP Pipeline configuration file.

Grab the default configuration file:

```
copy_snppipeline_data.py configurationFile
```

To run the SNP Pipeline with Bowtie2, edit `snppipeline.conf` with these settings:

```
SnpPipeline_Aligner="bowtie2"
Bowtie2Build_ExtraParams="" # substitute the command line options you want here
Bowtie2Align_ExtraParams="" # substitute the command line options you want here
```

To run the SNP Pipeline with Smalt, edit `snppipeline.conf` with these settings:

```
SnpPipeline_Aligner="smalt"
SmaltIndex_ExtraParams="" # substitute the command line options you want here
SmaltAlign_ExtraParams="" # substitute the command line options you want here
```

Then run the pipeline with the -c command line option:

```
run_snp_pipeline.sh -c snppipeline.conf -s mySamplesDir myReference.fasta
```

See also *Configuration*.

All-In-One SNP Pipeline Workflows

The sections below give detailed examples of workflows you can run with the all-in-one run_snp_pipeline.sh script.

All-In-One Workflow - Lambda Virus

The SNP Pipeline software distribution includes a small Lambda Virus data set that can be quickly processed to verify the basic functionality of the software.

Step 1 - Gather data:

```
# The SNP Pipeline distribution includes sample data organized as shown below:
snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq

# Copy the supplied test data to a work area:
cd test
copy_snppipeline_data.py lambdaVirusInputs testLambdaVirus
cd testLambdaVirus
```

Step 2 - Run the SNP Pipeline:

```
# Run the pipeline, specifing the locations of samples and the reference
#
# Specify the following options:
#   -s : samples parent directory
run_snp_pipeline.sh -s samples reference/lambda_virus.fasta
```

Step 3 - View and verify the results:

Upon successful completion of the pipeline, the snplist.txt file should have 165 entries, and the snplist_preserved.txt should have 136 entries. The SNP Matrix can be found in snpma.fasta and snpma_preserved.fasta. The corresponding reference bases are in the referenceSNP.fasta and referenceSNP_preserved.fasta:

```
# Verify the result files were created
ls -l snplist.txt
ls -l snpma.fasta
ls -l snpma.vcf
ls -l referenceSNP.fasta
ls -l snp_distance_matrix.tsv
ls -l snplist_preserved.txt
ls -l snpma_preserved.fasta
ls -l snpma_preserved.vcf
ls -l referenceSNP_preserved.fasta
ls -l snp_distance_matrix_preserved.tsv

# Verify correct results
copy_snppipeline_data.py lambdaVirusExpectedResults expectedResults
diff -q -s snplist.txt             expectedResults/snplist.txt
diff -q -s snpma.fasta             expectedResults/snpma.fasta
diff -q -s referenceSNP.fasta      expectedResults/referenceSNP.fasta
diff -q -s snp_distance_matrix.tsv expectedResults/snp_distance_matrix.tsv
diff -q -s snplist_preserved.txt             expectedResults/snplist_preserved.txt
diff -q -s snpma_preserved.fasta             expectedResults/snpma_preserved.fasta
diff -q -s referenceSNP_preserved.fasta      expectedResults/referenceSNP_preserved.
↪fasta
diff -q -s snp_distance_matrix_preserved.tsv expectedResults/snp_distance_matrix_
↪preserved.tsv

# View the per-sample metrics
xdg-open metrics.tsv
```

All-In-One Workflow - Salmonella Agona

The Salmonella Agona data set contains a small number of realistic sequences that can be processed in a reasonable amount of time. Due to the large size of real data, the sequences must be downloaded from the NCBI SRA. Follow the instructions below to download and process the data set.

Step 1 - Gather data:

```
# The SNP Pipeline distribution includes sample data organized as shown below:
snppipeline/data/agonaInputs/sha256sumCheck
snppipeline/data/agonaInputs/reference/NC_011149.fasta

# Copy the supplied test data to a work area:
mkdir testAgona
cd testAgona
copy_snppipeline_data.py agonaInputs cleanInputs
cd cleanInputs

# Create sample directories
mkdir -p samples/ERR178926  samples/ERR178927  samples/ERR178928  samples/ERR178929 ␣
↪samples/ERR178930

# Download sample data from SRA at NCBI. Note that we use the fastq-dump command from
#   the NCBI SRA-toolkit to fetch sample sequences. There are other ways to get the␣
↪data,
#   but the SRA-toolkit is easy to install, and does a good job of downloading large
#   files.
fastq-dump --split-files --outdir samples/ERR178926 ERR178926
fastq-dump --split-files --outdir samples/ERR178927 ERR178927
fastq-dump --split-files --outdir samples/ERR178928 ERR178928
fastq-dump --split-files --outdir samples/ERR178929 ERR178929
fastq-dump --split-files --outdir samples/ERR178930 ERR178930

# Check the data
#   The original data was used to generate a hash as follows:
#     sha256sum reference/*.fasta samples/*/*.fastq > sha256sumCheck
#   The command below checks the downloaded data (and the reference sequence) against␣
↪the
#     hashes that are saved in the sha256sumCheck file using sha256sum command, which␣
↪is
#     generally available on unix systems.
sha256sum -c sha256sumCheck
cd ..
```

Step 2 - Run the SNP Pipeline:

```
# Run the pipeline
# Specify the following options:
#   -m : mirror the input samples and reference files
#   -o : output directory
#   -s : samples parent directory
run_snp_pipeline.sh -m soft -o outputDirectory -s cleanInputs/samples cleanInputs/
↪reference/NC_011149.fasta
```

Step 3 - View and verify the results:

Upon successful completion of the pipeline, the snplist.txt file should have 3571 entries, and the snplist_preserved.txt should have 206 entries. The SNP Matrix can be found in snpma.fasta. The corresponding reference bases are in the files referenceSNP.fasta and referenceSNP_preserved.fasta:

```
# Verify the result files were created
ls -l outputDirectory/snplist.txt
ls -l outputDirectory/snpma.fasta
ls -l outputDirectory/snpma.vcf
ls -l outputDirectory/referenceSNP.fasta
ls -l outputDirectory/snp_distance_matrix.tsv
ls -l outputDirectory/snplist_preserved.txt
ls -l outputDirectory/snpma_preserved.fasta
ls -l outputDirectory/snpma_preserved.vcf
ls -l outputDirectory/referenceSNP_preserved.fasta
ls -l outputDirectory/snp_distance_matrix_preserved.tsv

# Verify correct results
copy_snppipeline_data.py agonaExpectedResults expectedResults
diff -q -s outputDirectory/snplist.txt             expectedResults/snplist.txt
diff -q -s outputDirectory/snpma.fasta             expectedResults/snpma.fasta
diff -q -s outputDirectory/referenceSNP.fasta      expectedResults/referenceSNP.fasta
diff -q -s outputDirectory/snp_distance_matrix.tsv expectedResults/snp_distance_
↪matrix.tsv
diff -q -s outputDirectory/snplist_preserved.txt            expectedResults/snplist_
↪preserved.txt
diff -q -s outputDirectory/snpma_preserved.fasta            expectedResults/snpma_
↪preserved.fasta
```

```
diff -q -s outputDirectory/referenceSNP_preserved.fasta      expectedResults/
↪referenceSNP_preserved.fasta
diff -q -s outputDirectory/snp_distance_matrix_preserved.tsv expectedResults/snp_
↪distance_matrix_preserved.tsv

# View the per-sample metrics
xdg-open outputDirectory/metrics.tsv
```

All-In-One Workflow - Listeria monocytogenes

This Listeria monocytogene data set is based on an oubreak investigation related to contamination in stone fruit. It only contains environmental/produce isolates, though the full investigation contained data obtained from clinical samples as well. Due to the large size of the data, the sequences must be downloaded from the NCBI SRA. The instructions below show how to create the data set and process it. We do the processing with the run_snp_pipeline.sh script, which does much of the work in one step, but provides less insight into (and control of) the analysis process.

This workflow illustrates how to run the SNP Pipeline on a High Performance Computing cluster (HPC) running the Torque job queue manager. If you do not have a cluster available, you can still work through this example – just remove the `-Q torque` command line option in step 2.

Step 1 - Create dataset:

```
# The SNP Pipeline distribution does not include the sample data, but does
#   include information about the sample data, as well as the reference
#   sequence.  The files are organized as shown below:
snppipeline/data/listeriaInputs/sha256sumCheck
snppipeline/data/listeriaInputs/reference/CFSAN023463.HGAP.draft.fasta
snppipeline/data/listeriaInputs/sampleList

# Copy the supplied test data to a work area:
mkdir testDir
cd testDir
copy_snppipeline_data.py listeriaInputs cleanInputs
cd cleanInputs

# Create sample directories and download sample data from SRA at NCBI. Note that
#   we use the fastq-dump command from the NCBI SRA-toolkit to fetch sample
#   sequences. There are other ways to get the data, but the SRA-toolkit is
#   easy to install, and does a good job of downloading large files.
mkdir samples
< sampleList xargs -I % sh -c ' mkdir samples/%; fastq-dump --split-files --outdir␣
↪samples/% %;'

# Check the data
#   The original data was used to generate a hash as follows:
#     sha256sum sampleList reference/*.fasta samples/*/*.fastq > sha256sumCheck
#   The command below checks the downloaded data (and the reference sequence) against␣
↪the
#     hashes that are saved in the sha256sumCheck file using sha256sum command, which␣
↪is
#     generally available on unix systems.
sha256sum -c sha256sumCheck
cd ..
```

Step 2 - Run the SNP Pipeline:

There are a couple of parameters you may need to adjust for this analysis or other analysis work that your do. These parameters are the number of CPU cores that are used, and the amount of memory that is used by the java virtual machine. Both can be set in a configuration file you can pass to run_snp_pipeline.sh with the `-c` option. See *Performance*.

Launch the pipeline:

```
# Run the pipeline.
# Specify the following options:
#   -m : mirror the input samples and reference files
#   -Q : HPC job queue manager
#   -o : output directory
#   -s : samples parent directory
run_snp_pipeline.sh -m soft -Q torque -o outputDirectory -s cleanInputs/samples␣
↪cleanInputs/reference/CFSAN023463.HGAP.draft.fasta
```

Step 3 - View and verify the results:

Upon successful completion of the pipeline, the snplist.txt file should have 11,502 entries, and the snplist_preserved.txt file should have 1,109 entries. The SNP Matrix can be found in snpma.fasta and snpma_preserved.fasta. The corresponding reference bases are in the referenceSNP.fasta and referenceSNP_preserved.fasta:

```
# Verify the result files were created
ls -l outputDirectory/snplist.txt
ls -l outputDirectory/snpma.fasta
ls -l outputDirectory/snpma.vcf
ls -l outputDirectory/referenceSNP.fasta
ls -l outputDirectory/snp_distance_matrix.tsv
ls -l outputDirectory/snplist_preserved.txt
ls -l outputDirectory/snpma_preserved.fasta
ls -l outputDirectory/snpma_preserved.vcf
ls -l outputDirectory/referenceSNP_preserved.fasta
ls -l outputDirectory/snp_distance_matrix_preserved.tsv

# Verify correct results
copy_snppipeline_data.py listeriaExpectedResults expectedResults
diff -q -s outputDirectory/snplist.txt             expectedResults/snplist.txt
diff -q -s outputDirectory/snpma.fasta             expectedResults/snpma.fasta
diff -q -s outputDirectory/referenceSNP.fasta      expectedResults/referenceSNP.fasta
diff -q -s outputDirectory/snp_distance_matrix.tsv expectedResults/snp_distance_
↪matrix.tsv
diff -q -s outputDirectory/snplist_preserved.txt             expectedResults/snplist_
↪preserved.txt
diff -q -s outputDirectory/snpma_preserved.fasta             expectedResults/snpma_
↪preserved.fasta
diff -q -s outputDirectory/referenceSNP_preserved.fasta      expectedResults/
↪referenceSNP_preserved.fasta
diff -q -s outputDirectory/snp_distance_matrix_preserved.tsv expectedResults/snp_
↪distance_matrix_preserved.tsv

# View the per-sample metrics
xdg-open outputDirectory/metrics.tsv
```

Step-by-Step Workflows

The run_snp_pipeline.sh script described above provides a simplified interface for running all the pipeline steps from a single command. If you need more control over the inputs, outputs, or processing steps, you can run the pipeline one step at a time.

The sections below give detailed examples of workflows you can run with the component tools of the pipeline.

Step-by-Step Workflow - Lambda Virus
Step-by-Step Workflow - Salmonella Agona
Step-by-Step Workflow - General Case

Step-by-Step Workflow - Lambda Virus

The SNP Pipeline software distribution includes a small Lambda Virus data set that can be quickly processed to verify the basic functionality of the software.

Step 1 - Gather data:

```
# The SNP Pipeline distribution includes sample data organized as shown below:
snppipeline/data/lambdaVirusInputs/reference/lambda_virus.fasta
snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample1/sample1_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample2/sample2_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample3/sample3_2.fastq
snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_1.fastq
snppipeline/data/lambdaVirusInputs/samples/sample4/sample4_2.fastq

# Copy the supplied test data to a work area:
cd test
copy_snppipeline_data.py lambdaVirusInputs testLambdaVirus
cd testLambdaVirus
```

Step 2 - Prep work:

```
# Create files of sample directories and fastQ files:
ls -d samples/* > sampleDirectories.txt
rm sampleFullPathNames.txt 2>/dev/null
cat sampleDirectories.txt | while read dir; do echo $dir/*.fastq >>␣
↪sampleFullPathNames.txt; done
# Determine the number of CPU cores in your computer
numCores=$(grep -c ^processor /proc/cpuinfo 2>/dev/null || sysctl -n hw.ncpu)
```

Step 3 - Prep the reference:

```
prepReference.sh reference/lambda_virus.fasta
```

Step 4 - Align the samples to the reference:

```
# Align each sample, one at a time, using all CPU cores
export Bowtie2Align_ExtraParams="--reorder -X 1000"
cat sampleFullPathNames.txt | xargs -n 2 -L 1 alignSampleToReference.sh reference/
↪lambda_virus.fasta
```

Step 5 - Prep the samples:

```
# Process the samples in parallel using all CPU cores
export VarscanMpileup2snp_ExtraParams="--min-var-freq 0.90"
cat sampleDirectories.txt | xargs -n 1 -P $numCores prepSamples.sh reference/lambda_
↪virus.fasta
```

Step 6 - Identify regions with abnormal SNP density and remove SNPs in these regions:

```
snp_filter.py -n var.flt.vcf sampleDirectories.txt reference/lambda_virus.fasta
```

Step 7 - Combine the SNP positions across all samples into the SNP list file:

```
create_snp_list.py -n var.flt.vcf -o snplist.txt sampleDirectories.txt
create_snp_list.py -n var.flt_preserved.vcf -o snplist_preserved.txt␣
↪sampleDirectories2.txt
```

Step 8 - Call the consensus base at SNP positions for each sample:

```
# Process the samples in parallel using all CPU cores
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist.txt --vcfFileName consensus.vcf -o XX/consensus.fasta XX/reads.all.pileup
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist_preserved.txt --vcfFileName consensus_preserved.vcf -o XX/consensus_
↪Preserved.fasta XX/reads.all.pileup
```

Step 9 - Create the SNP matrix:

```
create_snp_matrix.py -c consensus.fasta -o snpma.fasta sampleDirectories.txt
create_snp_matrix.py -c consensus_preserved.fasta -o snpma_preserved.fasta␣
↪sampleDirectories2.txt
```

Step 10 - Create the reference base sequence:

```
create_snp_reference_seq.py -l snplist.txt -o referenceSNP.fasta reference/lambda_
↪virus.fasta
create_snp_reference_seq.py -l snplist_preserved.txt -o referenceSNP_preserved.fasta␣
↪reference/lambda_virus.fasta
```

Step 11 - Collect metrics for each sample:

```
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX collectSampleMetrics.sh -o␣
↪XX/metrics XX reference/lambda_virus.fasta
```

Step 12 - Tabulate the metrics for all samples:

```
combineSampleMetrics.sh -n metrics -o metrics.tsv sampleDirectories.txt
```

Step 13 - Merge the VCF files for all samples into a multi-sample VCF file:

```
mergeVcf.sh -n consensus.vcf -o snpma.vcf sampleDirectories.txt
mergeVcf.sh -n consensus_preserved.vcf -o snpma_preserved.vcf sampleDirectories2.txt
```

Step 14 - Compute the SNP distances between samples:

```
calculate_snp_distances.py -p snp_distance_pairwise.tsv -m snp_distance_matrix.tsv␣
↪snpma.fasta
calculate_snp_distances.py -p snp_distance_pairwise_preserved.tsv -m snp_distance_
↪matrix_preserved.tsv snpma_preserved.fasta
```

Step 15 - View and verify the results:

Upon successful completion of the pipeline, the snplist.txt file should have 165 entries. The SNP Matrix can be found in snpma.fasta. The corresponding reference bases are in the referenceSNP.fasta file:

```
# Verify the result files were created
ls -l snplist.txt
ls -l snpma.fasta
ls -l snpma.vcf
ls -l referenceSNP.fasta
ls -l snp_distance_matrix.tsv
ls -l snplist_preserved.txt
ls -l snpma_preserved.fasta
ls -l snpma_preserved.vcf
ls -l referenceSNP_preserved.fasta
ls -l snp_distance_matrix_preserved.tsv

# Verify correct results
copy_snppipeline_data.py lambdaVirusExpectedResults expectedResults
diff -q -s snplist.txt             expectedResults/snplist.txt
diff -q -s snpma.fasta             expectedResults/snpma.fasta
diff -q -s referenceSNP.fasta      expectedResults/referenceSNP.fasta
diff -q -s snp_distance_matrix.tsv expectedResults/snp_distance_matrix.tsv
diff -q -s snplist_preserved.txt             expectedResults/snplist_preserved.txt
diff -q -s snpma_preserved.fasta             expectedResults/snpma_preserved.fasta
diff -q -s referenceSNP_preserved.fasta      expectedResults/referenceSNP_preserved.
↪fasta
diff -q -s snp_distance_matrix_preserved.tsv expectedResults/snp_distance_matrix_
↪preserved.tsv

# View the per-sample metrics
xdg-open metrics.tsv
```

Step-by-Step Workflow - Salmonella Agona

The Salmonella Agona data set contains realistic sequences that can be processed in a reasonable amount of time. Due to the large size of real data, the sequences must be downloaded from the NCBI SRA. Follow the instructions below to download and process the data set.

Step 1 - Gather data:

```
# The SNP Pipeline distribution includes sample data organized as shown below:
snppipeline/data/agonaInputs/sha256sumCheck
snppipeline/data/agonaInputs/reference/NC_011149.fasta

# Copy the supplied test data to a work area:
cd test
copy_snppipeline_data.py agonaInputs testAgona
cd testAgona

# Create sample directories
mkdir -p samples/ERR178926  samples/ERR178927  samples/ERR178928  samples/ERR178929 ␣
↪samples/ERR178930

# Download sample data from SRA at NCBI. Note that we use the fastq-dump command from
#   the NCBI SRA-toolkit to fetch sample sequences. There are other ways to get the␣
↪data,
#   but the SRA-toolkit is easy to install, and does a good job of downloading large
#   files.
fastq-dump --split-files --outdir samples/ERR178926 ERR178926
fastq-dump --split-files --outdir samples/ERR178927 ERR178927
fastq-dump --split-files --outdir samples/ERR178928 ERR178928
fastq-dump --split-files --outdir samples/ERR178929 ERR178929
fastq-dump --split-files --outdir samples/ERR178930 ERR178930

# Check the data
#   The original data was used to generate a hash as follows:
#     sha256sum reference/*.fasta samples/*/*.fastq > sha256sumCheck
#   The command below checks the downloaded data (and the reference sequence) against␣
↪the
#     hashes that are saved in the sha256sumCheck file using sha256sum command, which␣
↪is
#     generally available on unix systems.
sha256sum -c sha256sumCheck
```

Step 2 - Prep work:

```
# Create files of sample directories and fastQ files:
ls -d samples/* > sampleDirectories.txt
rm sampleFullPathNames.txt 2>/dev/null
cat sampleDirectories.txt | while read dir; do echo $dir/*.fastq >>␣
↪sampleFullPathNames.txt; done
# Determine the number of CPU cores in your computer
numCores=$(grep -c ^processor /proc/cpuinfo 2>/dev/null || sysctl -n hw.ncpu)
```

Step 3 - Prep the reference:

```
prepReference.sh reference/NC_011149.fasta
```

Step 4 - Align the samples to the reference:

```
# Align each sample, one at a time, using all CPU cores
export Bowtie2Align_ExtraParams="--reorder -X 1000"
cat sampleFullPathNames.txt | xargs -n 2 -L 1 alignSampleToReference.sh reference/NC_
↪011149.fasta
```

Step 5 - Prep the samples:

```
# Process the samples in parallel using all CPU cores
export VarscanMpileup2snp_ExtraParams="--min-var-freq 0.90"
cat sampleDirectories.txt | xargs -n 1 -P $numCores prepSamples.sh reference/NC_
↪011149.fasta
```

Step 6 - Identify regions with abnormal SNP density and remove SNPs in these regions:

```
snp_filter.py -n var.flt.vcf sampleDirectories.txt reference/NC_011149.fasta
```

Step 7 - Combine the SNP positions across all samples into the SNP list file:

```
create_snp_list.py -n var.flt.vcf -o snplist.txt sampleDirectories.txt
create_snp_list.py -n var.flt_preserved.vcf -o snplist_preserved.txt␣
↪sampleDirectories2.txt
```

Step 8 - Call the consensus base at SNP positions for each sample:

```
# Process the samples in parallel using all CPU cores
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist.txt --vcfFileName consensus.vcf -o XX/consensus.fasta XX/reads.all.pileup
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist_preserved.txt --vcfFileName consensus_preserved.vcf -o XX/consensus_
↪preserved.fasta XX/reads.all.pileup
```

Step 9 - Create the SNP matrix:

```
create_snp_matrix.py -c consensus.fasta -o snpma.fasta sampleDirectories.txt
create_snp_matrix.py -c consensus_preserved.fasta -o snpma_preserved.fasta␣
↪sampleDirectories2.txt
```

Step 10 - Create the reference base sequence:

```
create_snp_reference_seq.py -l snplist.txt -o referenceSNP.fasta reference/NC_011149.
↪fasta
create_snp_reference_seq.py -l snplist_preserved.txt -o referenceSNP_preserved.fasta␣
↪reference/NC_011149.fasta
```

Step 11 - Collect metrics for each sample:

```
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX collectSampleMetrics.sh -o␣
↪XX/metrics XX reference/NC_011149.fasta
```

Step 12 - Tabulate the metrics for all samples:

```
combineSampleMetrics.sh -n metrics -o metrics.tsv sampleDirectories.txt
```

Step 13 - Merge the VCF files for all samples into a multi-sample VCF file:

```
mergeVcf.sh -n consensus.vcf -o snpma.vcf sampleDirectories.txt
mergeVcf.sh -n consensus_preserved.vcf -o snpma_preserved.vcf sampleDirectories2.txt
```

Step 14 - Compute the SNP distances between samples:

```
calculate_snp_distances.py -p snp_distance_pairwise.tsv -m snp_distance_matrix.tsv␣
↪snpma.fasta
calculate_snp_distances.py -p snp_distance_pairwise_preserved.tsv -m snp_distance_
↪matrix_preserved.tsv snpma_preserved.fasta
```

Step 15 - View and verify the results:

Upon successful completion of the pipeline, the snplist.txt file should have 3623 entries. The SNP Matrix can be found in snpma.fasta. The corresponding reference bases are in the referenceSNP.fasta file:

```
# Verify the result files were created
ls -l snplist.txt
ls -l snpma.fasta
ls -l snpma.vcf
ls -l referenceSNP.fasta
ls -l snp_distance_matrix.tsv
ls -l snplist_preserved.txt
ls -l snpma_preserved.fasta
ls -l snpma_preserved.vcf
ls -l referenceSNP_preserved.fasta
ls -l snp_distance_matrix_preserved.tsv
```

```
# Verify correct results
copy_snppipeline_data.py agonaExpectedResults expectedResults
diff -q -s snplist.txt             expectedResults/snplist.txt
diff -q -s snpma.fasta             expectedResults/snpma.fasta
diff -q -s referenceSNP.fasta      expectedResults/referenceSNP.fasta
diff -q -s snp_distance_matrix.tsv expectedResults/snp_distance_matrix.tsv
diff -q -s snplist_preserved.txt            expectedResults/snplist_preserved.txt
diff -q -s snpma_preserved.fasta            expectedResults/snpma_preserved.fasta
diff -q -s referenceSNP_preserved.fasta      expectedResults/referenceSNP_preserved.
↪fasta
diff -q -s snp_distance_matrix_preserved.tsv expectedResults/snp_distance_matrix_
↪preserved.tsv

# View the per-sample metrics
xdg-open metrics.tsv
```

Step-by-Step Workflow - General Case

Step 1 - Gather data:

You will need the following data:

- Reference genome
- Fastq input files for multiple samples

Organize the data into separate directories for each sample as well as the reference. One possible directory layout is shown below. Note the mix of paired and unpaired samples:

```
./myProject/reference/my_reference.fasta
./myProject/samples/sample1/sampleA.fastq
./myProject/samples/sample2/sampleB.fastq
./myProject/samples/sample3/sampleC_1.fastq
./myProject/samples/sample3/sampleC_2.fastq
./myProject/samples/sample4/sampleD_1.fastq
./myProject/samples/sample4/sampleD_2.fastq
```

Step 2 - Prep work:

```
# Optional step: Copy your input data to a safe place:
cp -r myProject myProjectClean
# The SNP pipeline will generate additional files into the reference and sample␣
↪directories
cd myProject

# Create file of sample directories:
ls -d samples/* > sampleDirectories.txt

# get the *.fastq or *.fq files in each sample directory, possibly compresessed, on␣
↪one line per sample, ready to feed to bowtie
TMPFILE1=$(mktemp tmp.fastqs.XXXXXXXX)
cat sampleDirectories.txt | while read dir; do echo $dir/*.fastq* >> $TMPFILE1; echo
↪$dir/*.fq* >> $TMPFILE1; done
grep -v '*.fq*' $TMPFILE1 | grep -v '*.fastq*' > sampleFullPathNames.txt
rm $TMPFILE1

# Determine the number of CPU cores in your computer
numCores=$(grep -c ^processor /proc/cpuinfo 2>/dev/null || sysctl -n hw.ncpu)
```

```
```

Step 3 - Prep the reference:

```
prepReference.sh reference/my_reference.fasta
```

Step 4 - Align the samples to the reference:

```
# Align each sample, one at a time, using all CPU cores
export Bowtie2Align_ExtraParams="--reorder -X 1000"
cat sampleFullPathNames.txt | xargs -n 2 -L 1 alignSampleToReference.sh reference/my_
↪reference.fasta
```

Step 5 - Prep the samples:

```
# Process the samples in parallel using all CPU cores
export VarscanMpileup2snp_ExtraParams="--min-var-freq 0.90"
cat sampleDirectories.txt | xargs -n 1 -P $numCores prepSamples.sh reference/my_
↪reference.fasta
```

Step 6 - Identify regions with abnormal SNP density and remove SNPs in these regions:

```
snp_filter.py -n var.flt.vcf sampleDirectories.txt reference/my_reference.fasta
```

Step 7 - Combine the SNP positions across all samples into the SNP list file:

```
create_snp_list.py -n var.flt.vcf -o snplist.txt sampleDirectories.txt
create_snp_list.py -n var.flt_preserved.vcf -o snplist_preserved.txt␣
↪sampleDirectories2.txt
```

Step 8 - Call the consensus base at SNP positions for each sample:

```
# Process the samples in parallel using all CPU cores
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist.txt --vcfFileName consensus.vcf -o XX/consensus.fasta XX/reads.all.pileup
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX call_consensus.py -l␣
↪snplist_preserved.txt --vcfFileName consensus_preserved.vcf -o XX/consensus_
↪preserved.fasta XX/reads.all.pileup
```

Step 9 - Create the SNP matrix:

```
create_snp_matrix.py -c consensus.fasta -o snpma.fasta sampleDirectories.txt
create_snp_matrix.py -c consensus_preserved.fasta -o snpma_preserved.fasta␣
↪sampleDirectories2.txt
```

Step 10 - Create the reference base sequence:

```
# Note the .fasta file extension
create_snp_reference_seq.py -l snplist.txt -o referenceSNP.fasta reference/my_
↪reference.fasta
create_snp_reference_seq.py -l snplist_preserved.txt -o referenceSNP_preserved.fasta␣
↪reference/my_reference.fasta
```

Step 11 - Collect metrics for each sample:

```
cat sampleDirectories.txt | xargs -n 1 -P $numCores -I XX collectSampleMetrics.sh -o␣
↪XX/metrics XX reference/my_reference.fasta
```

Step 12 - Tabulate the metrics for all samples:

```
combineSampleMetrics.sh -n metrics -o metrics.tsv sampleDirectories.txt
```

Step 13 - Merge the VCF files for all samples into a multi-sample VCF file:

```
mergeVcf.sh -n consensus.vcf -o snpma.vcf sampleDirectories.txt
mergeVcf.sh -n consensus_preserved.vcf -o snpma_preserved.vcf sampleDirectories2.txt
```

Step 14 - Compute the SNP distances between samples:

```
calculate_snp_distances.py -p snp_distance_pairwise.tsv -m snp_distance_matrix.tsv␣
↪snpma.fasta
calculate_snp_distances.py -p snp_distance_pairwise_preserved.tsv -m snp_distance_
↪matrix_preserved.tsv snpma.fasta
```

Step 15 - View the results:

Upon successful completion of the pipeline, the snplist.txt identifies the SNP positions in all samples. The SNP Matrix can be found in snpma.fasta. The corresponding reference bases are in the referenceSNP.fasta file:

```
ls -l snplist.txt
ls -l snpma.fasta
ls -l snpma.vcf
ls -l referenceSNP.fasta
ls -l snp_distance_matrix.tsv
ls -l snplist_preserved.txt
ls -l snpma_preserved.fasta
ls -l snpma_preserved.vcf
ls -l referenceSNP_preserved.fasta
ls -l snp_distance_matrix_preserved.tsv

# View the per-sample metrics
xdg-open metrics.tsv
```

SNP Filtering

The SNP Pipeline removes abnormal SNPs from the ends of contigs and from regions where many SNPs are found in close proximity. The pipeline runs both ways, with SNP filtering, and without SNP filtering, generating pairs of output files. You can compare the output files to determine which positions were filtered. The filtered output files are named with the `_preserved` suffix, for example:

- snplist.txt : contains the unfiltered SNP positions with abnormal SNPs included
- snplist_preserved.txt : contains the filtered SNP positions without abnormal SNPs
- snpma.fasta : contains the unfiltered SNP matrix with abnormal SNPs included
- snpma_preserved.fasta : contains the filtered SNP matrix without abnormal SNPs

Other output files are named similarly.

The SNP filtering is performed by a script named `snp_filter.py`. It runs after the phase 1 SNP detection and impacts all subsequent processing steps. Abnormal regions are identified in each sample individually, and then SNPs in those regions are removed from *all* samples. Therefore, if you add or remove a sample from your analysis it may affect the final SNPs detected in all other samples. See *snp_filter.py*.

The sensitivity of the SNP filtering can be controlled with parameters in the configuration file by setting values in `RemoveAbnormalSnp_ExtraParams`. You can control the length of end-of-contig trimming, dense region window size, and maximum snps allowed within the window. See *Configuration*.

SNP Filtering With Outgroups

If there is an outgroup among the samples, you should configure the pipeline to exclude the outgroup samples from snp filtering. To exclude the outgroup samples:

First, make a file containing the sample ids of the outgroup samples, one sample id per line. The sample id is the name of the last subdirectory in the path to the sample:

```
SRR1556289
SRR1556294
```

Grab the default configuration file:

```
copy_snppipeline_data.py configurationFile
```

Edit `snppipeline.conf`, and change the `RemoveAbnormalSnp_ExtraParams` parameter:

```
Add the --out_group option with the path to the file containing the outgroup sample␣
↪ids.
```

Then run the snp pipeline with the -c command line options:

```
run_snp_pipeline.sh -c snppipeline.conf  -s mySamplesDir myReference.fasta
```

See also *Configuration*.

Excessive SNPs

Samples having many SNPs relative to the reference can slow the performance of the SNP Pipeline and greatly increase the size of the SNP matrix. The SNP Pipeline has the capability to exclude samples from processing when those samples have too many SNPs. This function excludes entire samples, not just regions within a sample. The samples with excessive SNPs exceeding a user-specified limit are excluded from the snp list, snp matrix, and snpma.vcf files.

There is also an indicator in the metrics file to identify the samples that have too many SNPs. A column in the metrics.tsv file, `Excluded_Sample`, indicates when a sample has been excluded from the snp matrix. This column is normally blank. See *Metrics*.

To exclude samples with excessive SNPs:

Grab the default configuration file:

```
copy_snppipeline_data.py configurationFile
```

Edit `snppipeline.conf`, and change this setting:

```
SnpPipeline_MaxSnps=1000  # substitute your threshold value here, or -1 to disable␣
↪this function
```

Then run the pipeline with the -c command line option:

```
run_snp_pipeline.sh -c snppipeline.conf -s mySamplesDir myReference.fasta
```

See also *Configuration*.

Metrics

After creating the SNP matrix, the pipeline collects and tabulates metrics for all of the samples. The metrics are first collected in one file per sample in the sample directories. A subsequent step combines the metrics for all the samples together into a single tab-separated file with one row per sample and one column per metric. The tabulated metrics file is named metrics.tsv by default.

The metrics are:

Column	Description
Sample	The name of the directory containing the sample fastq files.
Fastq Files	Comma separated list of fastq file names in the sample directory.
Fastq File Size	The sum of the sizes of the fastq files. This will be the compressed size if the files are compressed.
Machine	The sequencing instrument ID extracted from the compressed fastq.gz file header. If the fastq files are not compressed, the machine ID is not captured.
Flowcell	The flowcell used during the sequencing run, extracted from the compressed fastq.gz file header. If the fastq files are not compressed, the flowcell is not captured.
Number of Reads	The number of reads in the SAM file. When using paired fastq files, this number will be twice the number of reads reported by bowtie.
Percent of Reads Mapped	The percentage of reference-aligned reads in the SAM file.
Average Insert Size	The average observed template length of mapped paired reads as captured by SAMtools view TLEN value. This metric is not calculated for unpaired reads.
Average Pileup Depth	The average depth of coverage in the sample pileup file. This is calculated as the sum of the depth of the pileup across all pileup positions divided by the number of positions in the

Error Handling

The SNP Pipeline detects errors during execution and prevents execution of subsequent steps when earlier steps fail. A summary of errors is written to the `error.log` file. Detailed error messages are found in the log files for each process. See *Logging*.

By default, the SNP Pipeline is configured to stop when execution errors occur. However, it is possible some errors may affect only individual samples and other samples can still be processed. If you want the pipeline to continue processing after an error affecting only a single sample has occurred, you can try disabling the `SnpPipeline_StopOnSampleError` configuration parameter (not recommended). See *Configuration*. When `SnpPipeline_StopOnSampleError` is `false` the pipeline will attempt to continue subsequent processing steps when an error does not affect all samples. Errors are logged in the `error.log` file regardless of how the `SnpPipeline_StopOnSampleError` parameter is configured. You should review the `error.log` after running the pipeline to see a summary of any errors detected during execution.

Note: currently, when using the Torque job queue manager, the pipeline will always stop on errors regardless of the `SnpPipeline_StopOnSampleError` parameter setting.

When errors stop the execution of the pipeline on Grid Engine or Torque, other non-failing jobs in progress will continue until complete. However, subsequent job steps will not execute and instead will remain in the queue. On Grid Engine `qstat` will show output like the following:

```
3038927 0.55167 alignSampl app_sdavis   Eqw   01/15/2016 16:50:03
3038928 0.00000 prepSample app_sdavis   hqw   01/15/2016 16:50:04
3038929 0.00000 snpList    app_sdavis   hqw   01/15/2016 16:50:04
3038930 0.00000 callConsen app_sdavis   hqw   01/15/2016 16:50:04
3038931 0.00000 snpMatrix  app_sdavis   hqw   01/15/2016 16:50:04
3038932 0.00000 snpReferen app_sdavis   hqw   01/15/2016 16:50:04
3038933 0.00000 mergeVcf   app_sdavis   hqw   01/15/2016 16:50:05
3038934 0.00000 collectMet app_sdavis   hqw   01/15/2016 16:50:05
3038935 0.00000 combineMet app_sdavis   hqw   01/15/2016 16:50:05
```

To clear the jobs from the queue on Grid Engine:

```
seq 3038927 3038935 | xargs -I @ qdel @
```

CHAPTER 5

Correct and Reproducible Results

It is our goal to make the SNP Pipeline results both fully reproducible and as correct as current scientific understanding allows. As part of this effort, we document here problems we have found that have affected correctness. We also detail how we have built this software so the results are as reproducible as possible. In addition, we are building, collecting, and collating data sets that we use to assess both correctness and reproducibility of our software. As a project that is made possible only by very recent developments in science and technology, our efforts to ensure correctness and reproducibility are an ongoing effort. The publications we have produced in an effort to ensure scientific correctness are listed as references at the bottom of this document. This document will continue to evolve as we improve our process and as scientific advances occur.

Reproducible Results

We have made the SNP Pipeline results fully reproducible – not just the final SNP matrix, but each intermediate file as well. Reproducible results help us test and debug the pipeline and also facilitate collaborative efforts between researchers.

Public Availability

The SNP Pipeline source code is available on GitHub so anyone can download our source code. The GitHub repository contains some data sets that can be used to reproduce selected results. We also provide information on how to obtain and verify other data sets that we have used. (These data sets are large, so we do not provide them directly.)

Version Control

We use git internally for code development to ensure that we have control over our source code and can identify which version of code was used to produce any particular result. We tag/version commits of the code that we consider production releases and use them for the majority of our internal analyses. We also release each of these tagged versions to GitHub and to the Python Package Index for easy installation.

Parameters

The SNP pipeline behavior depends on the setting of a number of parameters that determine the behavior of various software packages that the pipeline uses. These parameters affect both the correctness and reproducibility of results. We have set all the parameters so that the results are reproducible. This entails setting seeds for all random number dependent processes, as well as specific choices for other parameters that can affect such behavior as the order of the results. We discuss these aspects of ensuring reproducibility in more detail in other portions of this document.

The pipeline depends on some fairly complex software packages, and these packages have large numbers of parameters. As released, the pipeline does not specify values for every possible parameter, but only those we have found it useful to modify in our work. A configuration file used by the pipeline provides a record of the parameters used and also makes it possible to customize the behavior of the pipeline by adding or modifying parameters as needed. Those wishing to further modify the software behavior will have to adjust the code to meet their needs.

We recommend retaining the parameter values used for any important results, ideally in a script or configuration file that is under version control. The pipeline generates log files documenting each run by capturing software versions and parameters used for each run.

Concurrency

The SNP Pipeline takes advantage of multiple CPU cores to run portions of the processing in parallel. However, concurrency can lead to non-deterministic behavior and different results when the pipeline is run repeatedly. The pipeline addresses known concurrency issues with bowtie and samtools.

Bowtie

The SNP Pipeline uses multiple CPU cores during the bowtie alignment. Unless told otherwise, when bowtie runs multiple concurrent threads, it generates output records in the SAM file in non-deterministic order. The consequence of this is the SAM files and Pileup files can differ between runs. This may appear as two adjacent read-bases swapped in the pileup files.

To work around this problem, the pipeline uses the `--reorder` bowtie command line option. The reorder option causes bowtie to generate output records in the same order as the reads in the input file. This is discussed in the bowtie documentation here: http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml#performance-options

SAMtools

The SNP Pipeline runs multiple samtools processes concurrently to generate pileups for each sample. When the samtools pileup process runs, it checks for the existence of the reference faidx file, `*.fai`. If the faidx file does not exist, samtools creates it automatically. However, multiple samtools mpileup processes can interfere with each other by attempting to create the file at the same time. This interference causes incorrect pipeline results.

To work around this problem, the pipeline explicitly creates the faidx file by running `samtools faidx` on the reference before running the mpileup processes. This prevents errors later when multiple samtools mpileup processes run concurrently.

Software Versions

Different versions of the software packages this pipeline uses can generate different results. This is important to be aware of if you end up comparing the results between runs. We share our observations from the versions of SAMtools and Bowtie that we have used below.

Bowtie

Different versions of Bowtie can generate different SAM files, which subsequently causes different pileups and different variant detection. For example, with our included data sets, Bowtie 2.1.0 and 2.2.2 produce functionally identical

SAM files when run on the Lambda Virus data set. However, the generated SAM files (and downstream results) are different when run on the Salmonella Agona data set.

SAMtools

SAMtools mpileup version 0.1.18 and version 0.1.19 differ in their default behavior. Version 0.1.19 can filter out bases with low quality, and by default, it excludes bases with quality score below 13 (95% accuracy). Version 0.1.18 does not have this capability, and thus different versions of SAMtools mpileup when run with the default parameters can produce different pileup files which can impact the snp list and snp matrix.

On one of our data sets with 116 samples, we observed these results:

- 36030 snps found when pileups generated with SAMtools 0.1.18
- 38154 snps found when pileups generated with SAMtools 0.1.19

Correct Results

As we have constructed our pipeline, we have found problems in our own software and in the various packages we use. To this point we have found one problem worth mentioning here.

SAMtools snp pileup difference from genome-wide pileup

An important processing step in the SNP Pipeline is creation of a pileup file per sample containing read pileups at the positions where snps were called in *any* of the samples. This pileup file should be a subset of the genome-wide pileup for each sample. However, the SAMtools software does not generate pileup records exactly matching the genome-wide pileup when given a list of positions for which the pileup should be generated. The differences are particularly evident at the first few snp positions and cause missing values in the SNP matrix. We first noticed this problem when the first or last position of the reference sequence was identified as a variant site. To work around this problem, the SNP Pipeline internally extracts the desired pileup records from the genome-wide pileup.

This SAMtools issue has been reported here: https://github.com/samtools/samtools/issues/282

Test Data Sets

We have created/curated a number of data sets for use in testing both the reproducibility and correctness of the pipeline. In the following sections we briefly describe these data sets.

Lambda Virus

This data set was built using the bowtie2 example, and intended to be a small test case and example that will run quickly and verify the basic functionality of the code.

Salmonella Agona

This data set was designed to contain realistic sequences, but not very many of them, so that it could be run in a reasonable amount of time. The data must be downloaded from the NCBI due to its large size. We provide a file of hashes that can easily be used to verify that the data downloaded matches the data originally used to produce our results. (Use sha256sum at the unix command line.)

Listeria monocytogenes

This is designed to be a realistic-sized data set based on an outbreak of L. m. in soft cheese. The data must be downloaded from the NCBI due to its large size. We provide a file of hashes that can easily be used to verify that the data downloaded matches the data originally used to produce our results. (Use sha256sum at the unix command line.)

Synthetic data sets

Coming soon in a future release

We are currently creating synthetic data sets based on simulating various evolutionary scenarios. The simulations are designed to be similar to what we would expect in the types of organisms we study (food-borne pathogens), with error structure appropriate for the platforms we use to do sequencing.

CHAPTER 6

FAQ / Troubleshooting Guide

Installation

Q: How can I avoid polluting my global python installation when installing the SNP pipeline?

A: You can either use a python virtual environment or install into your user area. To use a python virtual environment, see the *Get Started!* section for developers who want to contribute. To install into your user area instead of installing into your global site packages, do this:

```
$ pip install --user snp-pipeline
```

Q: The SNP Pipeline cannot find VarScan. How should I install it?

A: Download the VarScan jar file from SourceForge. Put the jar file anywhere. You need read-access to the jar file, but not execute-access. The supplied shell scripts expect the CLASSPATH environment variable to specify the path to the VarScan jar file. The CLASSPATH should include the filename, not just the directory. Define it like this in your .bashrc file:

```
export CLASSPATH=~/software/varscan.v2.3.9/VarScan.v2.3.9.jar:$CLASSPATH
```

Q: How can I uninstall the SNP pipeline?

A: If you installed with pip, you can uninstall from the command line:

```
$ pip uninstall snp-pipeline
```

Q: How can I rollback to an older version of the SNP pipeline?

A: You can revert to an older version with these commands:

```
$ pip uninstall snp-pipeline
$ pip install --user snp-pipeline==0.3.2  # substitute the version you want here
```

Q: Is there a way to install a specific release of the SNP pipeline from the github repository?

A: Yes, you can install a release from github with this command:

```
$ pip install --user https://github.com/CFSAN-Biostatistics/snp-pipeline/archive/v0.3.
↪2  # substitute the version you want here
```

Running the Pipeline

Q: Nothing works.

A: Make sure you have the proper dependencies on your path. Modify your path if necessary to include bowtie, samtools, and bcftools. See the question above about installing VarScan. In some cases, you may need to manually install Biopython. See the *Installation* section of this documentation.

Q: How can I verify the pipeline is installed and working properly?

A: The SNP Pipeline includes sets of test data with result files. You can run the pipeline against the test data to verify correct results. Follow the lambda virus workflow steps here: *All-In-One Workflow - Lambda Virus*.

Upon successful completion of the pipeline, the snplist.txt file should have 165 entries. The SNP Matrix can be found in snpma.fasta. It should have the following contents:

```
>sample1
AGCACCGGGACCCACGGCGCACGCAAAGATCCGAATTGCAGGGCGTACCTGGACCCCGGT
GACGGGGGATCGGGGACTCTTGGTGAGGAACTAAAACGAACATCCACGTTTTCATGGCGA
CTGCTTGCCAGGTGTCAGCACATTCCCTATATCGGTGGACACGTA
>sample2
GGCGCTAGGAGGCAAGCCTTGGTCGTGGTTAATAGTTACAAGGCGTGCGCGTACTGCCGT
CTCCTACTATCTCTGCCGCCTCTCGCGATCCGGACCGCAACACCAACTCTCTGGTGGCAT
CCTCTGAATCGTCGTGAGCATCTCAATTATATATTCGTCCGCGCG
>sample3
GGCGCTAGGAGGTACGCTTCGCGTGTGGATCAGCGCTACGGTGCCTATGCGTGACCCGCG
GAACTGGGTTCGCGTAAGGCAGTTCAGGTACGGCAACGTAGATCAAAGTTTAGAAACCAT
ACTCGTAATCCGCCTGACGCTACTCATTATGTATGTGGACGCCTG
>sample4
GAGGTTAACTGGCTCACCTCGCGCGTGTAACAGAGTAATAGGTTGAACGCCTACCCTGGT
GACCTGGGACGGCGGACGCCTGTTGAAGTAAGGAAACGATCCTAAGCGTCTTGATGGGAT
CCTATTAATCGGCGCGTGCATATTCATCGGACATGTCGAGGGGTG
```

Note: the expected pipeline results are also included in the distribution. To fetch the expected result files:

```
copy_snppipeline_data.py lambdaVirusExpectedResults myDirectoryForExpectedResults
```

After verifying correct results on the lambda data set, you can follow the workflow steps for the agona data set, *All-In-One Workflow - Salmonella Agona* or the listeria data set, *All-In-One Workflow - Listeria monocytogenes*. To fetch the expected result files:

```
copy_snppipeline_data.py agonaExpectedResults myDirectoryForExpectedResults
copy_snppipeline_data.py listeriaExpectedResults myDirectoryForExpectedResults
```

Q: My results for the included test data do not match the expected results. What is the cause?

A: Different versions of the executable tools can generate different results. The test data was generated with these versions:

- bowtie2 2.2.2
- samtools 1.3.1
- varscan 2.3.9

Q: How can I run the SNP Pipeline with a mix of paired and unpaired samples?

A: This is handled automatically if you use the run_snp_pipeline.sh script. If you are running alignSampleToReference.sh, run the script once per sample with either 1 fastq file or 2 fastq files. For example:

```
alignSampleToReference.sh  reference/NC_011149  samples/CFSAN000448/G0H235M04.RL10.
↪fastq
alignSampleToReference.sh  reference/NC_011149  samples/CFSAN000449/G00JH2D03.RL11.
↪fastq
alignSampleToReference.sh  reference/NC_011149  samples/CFSAN000450/HB4DJL101.RL1.
↪fastq
alignSampleToReference.sh  reference/NC_011149  samples/ERR178930/ERR178930_1.fastq ␣
↪samples/ERR178930/ERR178930_2.fastq
alignSampleToReference.sh  reference/NC_011149  samples/ERR178931/ERR178931_1.fastq ␣
↪samples/ERR178931/ERR178931_2.fastq
```

Q: How can I re-run some of the SNP Pipeline processing steps when I see a message that the results are already freshly built?

A: The SNP Pipeline detects freshly built result files and does not rebuild them. Result files are not rebuilt when the file timestamp is newer than all of the input files. To force a rebuild, specfify the `-f` option on the command line of any of the tools. To re-run only some of the steps, you can either delete the output files for that step or touch the input files for that step. All subsequent processing steps will also be re-run since their results will be out-of-date.

Q: How does the SNP Pipeline know which processing steps should be re-run after changing the configuration file?

A: It doesn't. If you change the configuration file, you may want to re-run some parts of the pipeline. The SNP Pipeline does not detect which parameters have changed since the last run. You must manually intervene to cause the pipeline to re-run the impacted processing steps. See the question above for guidance.

Q: What do the dashes ("-") in the snp matrix indicate?

A: Gaps, "-", are either missing bases (indels) or cases where there is insufficient information to make a consensus call (coverage depth too low, or consensus base frequency too low).

Q: Why are some snps missing from the snp matrix even when the snps were called by VarScan?

A: Older versions of VarScan failed to generate the header section of some VCF files. This in turn, caused the SNP Pipeline to ignore the first snp in the VCF file. Upgrade to a newer version VarScan.

Performance

Q: How can I control the number of concurrent processes lauched on my workstation?

A: If you are using a HPC with a job queue manager, the pipeline will automatically run multiple concurrent processes across multiple servers – there are no options to control the number of concurrent processes. On a workstation, the pipeline uses all available CPU cores by default and spawns multiple concurrent processes to use all the cores. However, you may want to control the number of concurrent processes. There are three steps in the pipeline where multiple processes are launched on a workstation. You can control the number of processes with the following parameters in the configuration file. These parameters are used only by the run_snp_pipeline.sh script:

```
# Maximum concurrent prepSamples.sh processes (SAMtools and Varscan)
MaxConcurrentPrepSamples=

# Maximum concurrent call_consensus.py processes
MaxConcurrentCallConsensus=
# Maximum concurrent collectSampleMetrics.sh processes
MaxConcurrentCollectSampleMetrics=
```

Q: How can I control the number of CPU cores used by the bowtie2 aligner?

A: By default, the SNP Pipeline will give bowtie2 all available CPU cores on a workstation and 8 CPU cores per sample on a high performance computing cluster. You can override the defaults with the `-p` bowtie2 option. Set the option either in the configuration file if you are running run_snp_pipeline.sh, or in the Bowtie2Align_ExtraParams environment variable if you are running alignSampleToReference.sh directly. For example, to run alignments with 16 concurrent threads:

```
Bowtie2Align_ExtraParams="--reorder -p 16"
```

On a workstation, alignments are run one at a time using multiple threads per alignment. On a cluster with a job queue, multiple alignments are run concurrently, each with multiple threads.

Q: How can I control the amount of memory that is used by the VarScan java virtual machine?

A: The amount of memory used by the java VM can be set by using the `-Xmx` java VM option. Set the option either in the configuration file if you are running run_snp_pipeline.sh, or in the VarscanJvm_ExtraParams environment variable if you are running prepSamples.sh directly. For example, to set maximum java heap size to 3000 MB:

```
VarscanJvm_ExtraParams="-Xmx3000m"
```

Developer Questions

Q: What causes "ImportError: No module named sphinx_rtd_theme" when building the documentation?

A: The documentation uses the *Read The Docs* theme. Install it like this:

```
$ pip install --user sphinx_rtd_theme
```

Q: I installed sphinx_rtd_theme, but I still get error "ImportError: No module named sphinx_rtd_theme".

A: Try running sphinx like this:

```
$ python /usr/bin/sphinx-build -b html  .  ./_build
```

Q: I changed one of the shell scripts, but the changes are ignored.

A: Reinstall the distribution. Do this:

```
$ python setup.py develop
```

CHAPTER 7

Configuration

You can customize the behavior of the SNP Pipeline by configuring parameters. Each step in the pipeline has a corresponding parameter allowing you to set one or more options for each tool in the pipeline.

Parameters can be configured either in a configuration file if you are using the `run_snp_pipeline.sh` script, or in environment variables.

The pipeline comes with a default configuration file. When you run the run_snp_pipeline.sh script without specifying a configuration file, it automatically uses the default supplied configuration file.

To get a copy of the default configuration file, run the following command. This will create a file called `snppipeline.conf`:

```
copy_snppipeline_data.py configurationFile
```

To customize the pipeline behavior, edit the configuration file and pass the file to the run_snp_pipeline.sh script:

```
run_snp_pipeline.sh -c snppipeline.conf ...
```

When the run_snp_pipeline.sh script runs, it copies the configuration file to the log directory for the run, capturing the configuration used for each run.

If you decide not to use the run_snp_pipeline.sh script, you can still customize the behavior of the pipeline tools, but you will need to set (and export) environment variables with the same names as the parameters in the configuration file.

The available configuration parameters are described below.

SnpPipeline_StopOnSampleError

Controls whether the pipeline exits upon detecting errors affecting only a single sample. The pipeline will always stop upon detecting global errors affecting all samples.

Default:

When this parameter is not set to a value, the pipeline will stop upon detecting single sample errors. If you want the pipeline to continue, you must explicitly set this parameter false.

Example:

```
SnpPipeline_StopOnSampleError=false
```

MaxConcurrentPrepSamples

Controls the number of prepSamples.sh (SAMtools and Varscan) processes running concurrently on a workstation. This parameter is ignored when running the pipeline on an HPC job queue. This parameter is used by run_snp_pipeline.sh only.

Default:

When this parameter is not set to a value, the pipeline will launch multiple concurrent processes using all available CPU cores on a workstation.

Example:

```
MaxConcurrentPrepSamples=2
```

MaxConcurrentCallConsensus

Controls the number of call_consensus.py processes running concurrently on a workstation. This parameter is ignored when running the pipeline on an HPC job queue. This parameter is used by run_snp_pipeline.sh only.

Default:

When this parameter is not set to a value, the pipeline will launch multiple concurrent processes using all available CPU cores on a workstation.

Example:

```
MaxConcurrentCallConsensus=4
```

MaxConcurrentCollectSampleMetrics

Controls the number of collectSampleMetrics.sh processes running concurrently on a workstation. This parameter is ignored when running the pipeline on an HPC job queue. This parameter is used by run_snp_pipeline.sh only.

Default:

When this parameter is not set to a value, the pipeline will launch multiple concurrent processes using all available CPU cores on a workstation.

Example:

```
MaxConcurrentCollectSampleMetrics=4
```

SnpPipeline_MaxSnps

Controls the maximum number of snps allowed for each sample. Any sample with excessive snps exceeding this limit will be excluded from the snp list, snp matrix, and snpma.vcf file. When set to -1, this parameter is disabled.

Default:

Do not leave this parameter unset. To disable the excessive snp filtering and include all samples regardless of the number of snps, set the parameter to -1

Example:

```
SnpPipeline_MaxSnps=1000
```

SnpPipeline_Aligner

Controls which reference-based aligner is used to map reads to the reference genome. The choices are `bowtie2` or `smalt`.

Default:

When this parameter is not set to a value, the pipeline will use the bowtie2 aligner.

Example:

```
SnpPipeline_Aligner="smalt"
```

Bowtie2Build_ExtraParams

Specifies options passed to the bowtie2 indexer. Any of the bowtie2-build options can be specified.

Default: none

Example:

```
Bowtie2Build_ExtraParams="--offrate 3"
```

SmaltIndex_ExtraParams

Specifies options passed to the smalt indexer. Any of the smalt index options can be specified.

Default: none

Example:

```
SmaltIndex_ExtraParams="-k 20 -s 1"
```

SamtoolsFaidx_ExtraParams

Specifies options passed to the SAMtools faidx indexer. Any of the SAMtools faidx options can be specified.

Default: none

Example:

```
SamtoolsFaidx_ExtraParams=""
```

Bowtie2Align_ExtraParams

Specifies options passed to the bowtie2 aligner. Any of the bowtie2 aligner options can be specified.

Default:

If you do not specify the `-p` option, it defaults to 8 threads on an HPC or all cpu cores otherwise.
    There is no way to completely suppress the -p option.
If Bowtie2Align_ExtraParams is not set to any value, the `--reorder` option is enabled by default.
    Any value, even a single space, will suppress this default option.

Parameter Notes:

`-p` : bowtie2 uses the specified number of parallel search threads
`--reorder` : generate output records in the same order as the reads in the input file
`-X` : maximum inter-mate fragment length for valid concordant paired-end alignments

Example:

```
Bowtie2Align_ExtraParams="--reorder -p 16 -X 1000"
```

SmaltAlign_ExtraParams

Specifies options passed to the smalt mapper. Any of the smalt map options can be specified.

Default:

If you do not specify the `-n` option, it defaults to 8 threads on an HPC or all cpu cores otherwise.
    There is no way to completely suppress the -n option.
If SmaltAlign_ExtraParams is not set to any value, the `-O` option is enabled by default.
    Any value, even a single space, will suppress this default option.

Parameter Notes:

`-n` : number of parallel alignment threads
`-O` : generate output records in the same order as the reads in the input file
`-i` : maximum insert size for paired-end reads
`-r` : random number seed, if seed < 0 reads with multiple best mappings are reported as 'not mapped'
`-y` : filters output alignments by a threshold in the number of exactly matching nucleotides

Example:

```
SmaltAlign_ExtraParams="-O -i 1000 -r 1"
```

SamtoolsSamFilter_ExtraParams

Specifies options passed to the SAMtools view tool when filtering the SAM file. Any of the SAMtools view options can be specified.

Default:

If SamtoolsSamFilter_ExtraParams is not set, the "-F 4" option is enabled by default.
Any value, even a single space, will suppress the -F option.

Parameter Notes:

`-F 4` : discard unmapped reads

Example:

```
SamtoolsSamFilter_ExtraParams="-F 4"
```

SamtoolsSort_ExtraParams

Specifies options passed to the SAMtools sort tool when sorting the BAM file. Any of the SAMtools sort options can be specified.

Default: None

Example:

```
SamtoolsSort_ExtraParams=""
```

SamtoolsMpileup_ExtraParams

Specifies options passed to the SAMtools mpileup tool. Any of the SAMtools mpileup options can be specified.

Default: None

Parameter Notes:

`-q` : minimum mapping quality for an alignment to be used
`-Q` : minimum base quality for a base to be considered
`-x` : disable read-pair overlap detection

Example:

```
SamtoolsMpileup_ExtraParams="-q 0 -Q 13"
```

VarscanMpileup2snp_ExtraParams

Specifies options passed to the Varscan mpileup2snp tool. Any of the Varscan mpileup2snp options can be specified.

Default: None

Parameter Notes:

`--min-avg-qual` : minimum base quality at a position to count a read
`--min-var-freq` : minimum variant allele frequency threshold

Example:

```
VarscanMpileup2snp_ExtraParams="--min-avg-qual 15 --min-var-freq 0.90"
```

VarscanJvm_ExtraParams

Specifies options passed to the Varscan Java Virtual Machine. Any of the JVM options can be specified.

Default: None

Parameter Notes:

`-Xmx300m` : use 300 MB memory (modify as needed)

Example:

```
VarscanJvm_ExtraParams="-Xmx300m"
```

RemoveAbnormalSnp_ExtraParams

Specifies options passed to the snp_filter.py script.

Default: None

Parameter Notes:

`--edge_length` The length of the edge regions in a contig, in which all SNPs will be removed.

`--window_size` The length of the window in which the number of SNPs should be no more than max_num_snp.

`--max_snp` The maximum number of SNPs allowed in a window.

`--out_group` Relative or absolute path to the file indicating outgroup samples, one sample ID per line.

Example:

```
RemoveAbnormalSnp_ExtraParams="--edge_length 500 --window_size 1000 --max_snp 3 --out_
↪group /path/to/outgroupSamples.txt"
```

CreateSnpList_ExtraParams

Specifies options passed to create_snp_list.py.

Default: None

Example:

```
CreateSnpList_ExtraParams="--verbose 1"
```

CallConsensus_ExtraParams

Specifies options passed to call_consensus.py.

Default: None

Parameter Notes:

`--minBaseQual` Mimimum base quality score to count a read. All other snp filters take effect after the low-quality reads are discarded.

`--minConsFreq` Consensus frequency. Mimimum fraction of high-quality reads supporting the consensus to make a call.

`--minConsStrdDpth` Consensus strand depth. Minimum number of high-quality reads supporting the consensus which must be present on both the forward and reverse strands to make a call

`--minConsStrdBias` Strand bias. Minimum fraction of the high-quality consensus-supporting reads which must be present on both the forward and reverse strands to make a call. The numerator of this fraction is the number of high-quality consensus-supporting reads on one strand. The denominator is the total number of high-quality consensus-supporting reads on both strands combined.

--vcfFileName VCF Output file name. If specified, a VCF file with this file name will be created in the same directory as the consensus fasta file for this sample.

--vcfAllPos Flag to cause VCF file generation at all positions, not just the snp positions. This has no effect on the consensus fasta file, it only affects the VCF file. This capability is intended primarily as a diagnostic tool and enabling this flag will greatly increase execution time.

--vcfPreserveRefCase Flag to cause the VCF file generator to emit each reference base in uppercase/lowercase as it appears in the reference sequence file. If not specified, the reference bases are emitted in uppercase.

Example:

```
CallConsensus_ExtraParams="--verbose 1 --minBaseQual 15 --vcfFileName consensus.vcf"
```

CreateSnpMatrix_ExtraParams

Specifies options passed to create_snp_matrix.py.

Default: None

Example:

```
CreateSnpMatrix_ExtraParams="--verbose 1"
```

CreateSnpReferenceSeq_ExtraParams

Specifies options passed to create_snp_reference_seq.py.

Default: None

Example:

```
CreateSnpReferenceSeq_ExtraParams="--verbose 1"
```

MergeVcf_ExtraParams

Specifies options passed to mergeVcf.sh

Default: none

Example:

```
MergeVcf_ExtraParams="-n sample.vcf"
```

CollectSampleMetrics_ExtraParams

Specifies options passed to collectSampleMetrics.sh

Default: none

Example:

```
CollectSampleMetrics_ExtraParams="-v consensus.vcf"
```

CombineSampleMetrics_ExtraParams

Specifies options passed to combineSampleMetrics.sh

Default: none

Parameter Notes:

`-s` : Emit column headings with spaces instead of underscores

Example:

```
CombineSampleMetrics_ExtraParams="-s"
```

Torque_StripJobArraySuffix

Controls stripping the suffix from the job id when specifying Torque job array dependencies. It may be necessary to change this parameter if run_snp_pipeline.sh fails with an illegal qsub dependency error.

Example:

```
Torque_StripJobArraySuffix=false
```

GridEngine_StripJobArraySuffix

Controls stripping the suffix from the job id when specifying Grid Engine job array dependencies. It may be necessary to change this parameter if run_snp_pipeline.sh fails with an illegal qsub dependency error.

Example:

```
GridEngine_StripJobArraySuffix=true
```

GridEngine_PEname

Specifies the name of the Grid Engine parallel environment. This is only needed when running the SNP Pipeline on a High Performance Computing cluster with the Grid Engine job manager. Contact your HPC system administrator to determine the name of your parallel environment. Note: the name of this parameter was PEname in releases prior to 0.4.0.

Example:

```
GridEngine_PEname="mpi"
```

GridEngine_QsubExtraParams

Specifies extra options passed to qsub when running the SNP Pipeline on the Grid Engine job scheduler.

Default: None

Example:

```
GridEngine_QsubExtraParams="-q bigmem.q"
```

Torque_QsubExtraParams

Specifies extra options passed to qsub when running the SNP Pipeline on the Torque job scheduler.

Default: None

Example:

```
Torque_QsubExtraParams="-l pmem=16gb"
```

CHAPTER 8

Command Reference

copy_snppipeline_data.py

```
usage: copy_snppipeline_data.py [-h] whichData [destDirectory]

Copy SNP Pipeline data to a specified directory.

positional arguments:
  whichData           Which of the supplied data sets to copy.  The choices are:
                          lambdaVirusInputs           : Input reference and fastq files
                          lambdaVirusExpectedResults : Expected results files
                          agonaInputs                 : Input reference file
                          agonaExpectedResults        : Expected results files
                          listeriaInputs              : Input reference file
                          listeriaExpectedResults     : Expected results files
                          configurationFile           : File of parameters to customize␣
↪the
                                                        SNP pipeline

                      Note: the lambda virus data set is complete with input data and␣
↪expected
                      results.  The agona and listeria data sets have the reference␣
↪genome and
                      the expected results, but not the input fastq files, because the␣
↪files are
                      too large to include with the package.  (default: None)

  destDirectory       Destination directory into which the SNP pipeline data files␣
↪will be copied.
                      The data files are copied into the destination directory if the␣
↪directory
                      already exists.  Otherwise the destination directory is created␣
↪and the
                      data files are copied there.  (default: current directory)
```

```
optional arguments:
  -h, --help     show this help message and exit

Example:
# create a new directory "testLambdaVirus" and copy the input data there
$ copy_snppipeline_data.py lambdaVirusInputs testLambdaVirus
```

run_snp_pipeline.sh

```
usage: run_snp_pipeline.sh [-h] [-f] [-m MODE] [-c FILE] [-Q torque|grid] [-o DIR] (-
↪s DIR|-S FILE)
                           referenceFile

Run the SNP Pipeline on a specified data set.

Positional arguments:
  referenceFile  : Relative or absolute path to the reference fasta file.

Options:
  -h             : Show this help message and exit.

  -f             : Force processing even when result files already exist and
                   are newer than inputs.

  -m MODE        : Create a mirror copy of the reference directory and all the sample
                   directories.  Use this option to avoid polluting the reference␣
↪directory and
                   sample directories with the intermediate files generated by the␣
↪snp pipeline.
                   A "reference" subdirectory and a "samples" subdirectory are␣
↪created under
                   the output directory (see the -o option).  One directory per␣
↪sample is created
                   under the "samples" directory.  Three suboptions allow a choice of␣
↪how the
                   reference and samples are mirrored.
                     -m soft : creates soft links to the fasta and fastq files␣
↪instead of copying
                     -m hard : creates hard links to the fasta and fastq files␣
↪instead of copying
                     -m copy : copies the fasta and fastq files

  -c FILE        : Relative or absolute path to a configuration file for overriding␣
↪defaults
                   and defining extra parameters for the tools and scripts within the␣
↪pipeline.
                   Note: A default parameter configuration file named "snppipeline.
↪conf" is
                         used whenever the pipeline is run without the -c option.  The
                         configuration file used for each run is copied into the log␣
↪directory,
                         capturing the parameters used during the run.
  -Q torque|grid : Job queue manager for remote parallel job execution in an HPC␣
↪environment.
                   Currently "torque" and "grid" are supported.  If not specified,␣
↪the pipeline
                   will execute locally.

  -o DIR         : Output directory for the snp list, snp matrix, and reference snp␣
↪files.
                   Additional subdirectories are automatically created under the␣
↪output
                   directory for logs files and the mirrored samples and reference␣
↪files
                   (see the -m option).  The output directory will be created if it␣
↪does
                   not already exist.  If not specified, the output files are written␣
↪to
                   the current working directory.  If you re-run the pipeline on␣
↪previously
                   processed samples, and specify a different output directory, the
                   pipeline will not rebuild everything unless you either force a␣
↪rebuild
                   (see the -f option) or you request mirrored inputs (see the -m␣
↪option).

  -s DIRECTORY   : Relative or absolute path to the parent directory of all the sample
                   directories.  The -s option should be used when all the sample␣
↪directories
                   are in subdirectories immediately below a parent directory.
                   Note: You must specify either the -s or -S option, but not both.
                   Note: The specified directory should contain only a collection of␣
↪sample
                         directories, nothing else.
                   Note: Unless you request mirrored inputs, see the -m option,␣
↪additional files
                         will be written to each of the sample directories during the␣
↪execution
                         of the SNP Pipeline

  -S FILE        : Relative or absolute path to a file listing all of the sample␣
↪directories.
                   The -S option should be used when the samples are not under a␣
↪common parent
                   directory.
                   Note: If you are not mirroring the samples (see the -m option),␣
↪you can
                         improve parallel processing performance by sorting the the␣
↪list of
                         directories descending by size, largest first.  The -m option
                         automatically generates a sorted directory list.
                   Note: You must specify either the -s or -S option, but not both.
                   Note: Unless you request mirrored inputs, see the -m option,␣
↪additional files
                         will be written to each of the sample directories during the␣
↪execution
                         of the SNP Pipeline
```

prepReference.sh

```
usage: prepReference.sh [-h] [-f] referenceFile

Index the reference genome for subsequent alignment, and create
the faidx index file for subsequent pileups. The output is written
to the reference directory.

Positional arguments:
  referenceFile    : Relative or absolute path to the reference fasta file

Options:
  -h               : Show this help message and exit
  -f               : Force processing even when result files already exist and
                     are newer than inputs
```

alignSampleToReference.sh

```
usage: alignSampleToReference.sh [-h] [-f] referenceFile sampleFastqFile1␣
↪[sampleFastqFile2]

Align the sequence reads for a specified sample to a specified reference genome.
The output is written to the file "reads.sam" in the sample directory.

Positional arguments:
  referenceFile    : Relative or absolute path to the reference fasta file
  sampleFastqFile1 : Relative or absolute path to the fastq file
  sampleFastqFile2 : Optional relative or absolute path to the mate fastq file, if␣
↪paired

Options:
  -h               : Show this help message and exit
  -f               : Force processing even when result files already exist and
                     are newer than inputs
```

prepSamples.sh

```
usage: prepSamples.sh [-h] [-f] referenceFile sampleDir

Find variants in a specified sample.
The output files are written to the sample directory.

Positional arguments:
  referenceFile    : Relative or absolute path to the reference fasta file
  sampleDir        : Relative or absolute directory of the sample

Options:
  -h               : Show this help message and exit
  -f               : Force processing even when result files already exist and
                     are newer than inputs
```

snp_filter.py

```
usage: snp_filter.py [-h] [-f] [-n NAME] [-l EDGE_LENGTH] [-w WINDOW_SIZE]
                     [-m MAX_NUM_SNPs] [-g OUT_GROUP] [-v 0..5] [--version]
                     sampleDirsFile refFastaFile

Remove abnormally dense SNPs from the input VCF file, save the reserved SNPs
into a new VCF file, and save the removed SNPs into another VCF file.

positional arguments:
  sampleDirsFile        Relative or absolute path to file containing a list of
                        directories -- one per sample
  refFastaFile          Relative or absolute path to the reference fasta file

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result files already exist
                        and are newer than inputs (default: False)
  -n NAME, --vcfname NAME
                        File name of the input VCF files which must exist in
                        each of the sample directories (default: var.flt.vcf)
  -l EDGE_LENGTH, --edge_length EDGE_LENGTH
                        The length of the edge regions in a contig, in which
                        all SNPs will be removed. (default: 500)
  -w WINDOW_SIZE, --window_size WINDOW_SIZE
                        The length of the window in which the number of SNPs
                        should be no more than max_num_snp. (default: 1000)
  -m MAX_NUM_SNPs, --max_snp MAX_NUM_SNPs
                        The maximum number of SNPs allowed in a window.
                        (default: 3)
  -g OUT_GROUP, --out_group OUT_GROUP
                        Relative or absolute path to the file indicating
                        outgroup samples, one sample ID per line. (default:
                        None)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

create_snp_list.py

```
usage: create_snp_list.py [-h] [-f] [-n NAME] [--maxsnps INT] [-o FILE]
                          [-v 0..5] [--version]
                          sampleDirsFile filteredSampleDirsFile

Combine the SNP positions across all samples into a single unified SNP list
file identifing the postions and sample names where SNPs were called.

positional arguments:
  sampleDirsFile        Relative or absolute path to file containing a list of
                        directories -- one per sample
  filteredSampleDirsFile
                        Relative or absolute path to the output file that will
                        be created containing the filtered list of sample
                        directories -- one per sample. The samples in this
                        file are those without an excessive number of snps.
                        See the --maxsnps parameter.

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs (default: False)
  -n NAME, --vcfname NAME
                        File name of the VCF files which must exist in each of
                        the sample directories (default: var.flt.vcf)
  --maxsnps INT         Exclude samples having more than this maximum allowed
                        number of SNPs. Set to -1 to disable this function.
                        (default: -1)
  -o FILE, --output FILE
                        Output file. Relative or absolute path to the SNP list
                        file (default: snplist.txt)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

create_snp_pileup.py

```
usage: create_snp_pileup.py [-h] [-f] [-l FILE] [-a FILE] [-o FILE] [-v 0..5]
                            [--version]

Create the SNP pileup file for a sample -- the pileup file at the positions
where SNPs were called in any of the samples.

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs (default: False)
  -l FILE, --snpListFile FILE
                        Relative or absolute path to the SNP list file across
                        all samples (default: snplist.txt)
  -a FILE, --allPileupFile FILE
                        Relative or absolute path to the genome-wide pileup
                        file for this sample (default: reads.all.pileup)
  -o FILE, --output FILE
                        Output file. Relative or absolute path to the sample
                        SNP pileup file (default: reads.snp.pileup)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

call_consensus.py

```
usage: call_consensus.py [-h] [-f] [-l FILE] [-e FILE] [-o FILE] [-q INT]
                         [-c FREQ] [-d INT] [-b FREQ] [--vcfFileName NAME]
                         [--vcfRefName NAME] [--vcfAllPos]
                         [--vcfPreserveRefCase] [--vcfFailedSnpGt {.,0,1}]
                         [-v 0..5] [--version]
                         allPileupFile
```

```

Call the consensus base for a sample at the specified positions where SNPs
were previously called in any of the samples. Generates a single-sequence
fasta file with one base per specified position.

positional arguments:
  allPileupFile         Relative or absolute path to the genome-wide pileup
                        file for this sample.

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs. (default: False)
  -l FILE, --snpListFile FILE
                        Relative or absolute path to the SNP list file across
                        all samples. (default: snplist.txt)
  -e FILE, --excludeFile FILE
                        VCF file of positions to exclude. (default: None)
  -o FILE, --output FILE
                        Output file. Relative or absolute path to the
                        consensus fasta file for this sample. (default:
                        consensus.fasta)
  -q INT, --minBaseQual INT
                        Mimimum base quality score to count a read. All other
                        snp filters take effect after the low-quality reads
                        are discarded. (default: 0)
  -c FREQ, --minConsFreq FREQ
                        Consensus frequency. Mimimum fraction of high-quality
                        reads supporting the consensus to make a call.
                        (default: 0.6)
  -d INT, --minConsStrdDpth INT
                        Consensus strand depth. Minimum number of high-quality
                        reads supporting the consensus which must be present
                        on both the forward and reverse strands to make a
                        call. (default: 0)
  -b FREQ, --minConsStrdBias FREQ
                        Strand bias. Minimum fraction of the high-quality
                        consensus-supporting reads which must be present on
                        both the forward and reverse strands to make a call.
                        The numerator of this fraction is the number of high-
                        quality consensus-supporting reads on one strand. The
                        denominator is the total number of high-quality
                        consensus-supporting reads on both strands combined.
                        (default: 0)
  --vcfFileName NAME    VCF Output file name. If specified, a VCF file with
                        this file name will be created in the same directory
                        as the consensus fasta file for this sample. (default:
                        None)
  --vcfRefName NAME     Name of the reference file. This is only used in the
                        generated VCF file header. (default: Unknown
                        reference)
  --vcfAllPos           Flag to cause VCF file generation at all positions,
                        not just the snp positions. This has no effect on the
                        consensus fasta file, it only affects the VCF file.
                        This capability is intended primarily as a diagnostic
                        tool and enabling this flag will greatly increase
                        execution time. (default: False)
  --vcfPreserveRefCase  Flag to cause the VCF file generator to emit each
```

```
                        reference base in uppercase/lowercase as it appears in
                        the reference sequence file. If not specified, the
                        reference base is emitted in uppercase. (default:
                        False)
  --vcfFailedSnpGt {.,0,1}
                        Controls the VCF file GT data element when a snp fails
                        filters. Possible values: .) The GT element will be a
                        dot, indicating unable to make a call. 0) The GT
                        element will be 0, indicating the reference base. 1)
                        The GT element will be the ALT index of the most
                        commonly occuring base, usually 1. (default: .)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

mergeVcf.sh

```
usage: mergeVcf.sh [-h] [-f] [-n NAME] [-o FILE] sampleDirsFile

Merge the vcf files from all samples into a single multi-vcf file for all samples.

Before running this command, the vcf file for each sample must be created by the
call_consensus.py script.

Positional arguments:
  sampleDirsFile    : Relative or absolute path to file containing a list of
                      directories -- one per sample

Options:
  -h                : Show this help message and exit
  -f                : Force processing even when result files already exist and
                      are newer than inputs
  -n NAME           : File name of the vcf files which must exist in each of
                      the sample directories. (default: consensus.vcf)
  -o FILE           : Output file. Relative or absolute path to the merged
                      multi-vcf file. (default: snpma.vcf)
```

create_snp_matrix.py

```
usage: create_snp_matrix.py [-h] [-f] [-c NAME] [-o FILE] [-v 0..5]
                            [--version]
                            sampleDirsFile

Create the SNP matrix containing the consensus base for each of the samples at
the positions where SNPs were called in any of the samples. The matrix
contains one row per sample and one column per SNP position. Non-SNP positions
are not included in the matrix. The matrix is formatted as a fasta file, with
each sequence (all of identical length) corresponding to the SNPs in the
correspondingly named sequence.

positional arguments:
  sampleDirsFile        Relative or absolute path to file containing a list of
```

```
                        directories -- one per sample

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs (default: False)
  -c NAME, --consFileName NAME
                        File name of the previously created consensus SNP call
                        file which must exist in each of the sample
                        directories (default: consensus.fasta)
  -o FILE, --output FILE
                        Output file. Relative or absolute path to the SNP
                        matrix file (default: snpma.fasta)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

calculate_snp_distances.py

```
usage: calculate_snp_distances.py [-h] [-f] [-p FILE] [-m FILE] [-v 0..5]
                                  [--version]
                                  snpMatrixFile

Calculate pairwise SNP distances from the multi-fasta SNP matrix. Generates a
file of pairwise distances and a file containing a matrix of distances.

positional arguments:
  snpMatrixFile         Relative or absolute path to the input multi-fasta SNP
                        matrix file.

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs (default: False)
  -p FILE, --pairs FILE
                        Relative or absolute path to the pairwise distance
                        output file. (default: None)
  -m FILE, --matrix FILE
                        Relative or absolute path to the distance matrix
                        output file. (default: None)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

create_snp_reference_seq.py

```
usage: create_snp_reference_seq.py [-h] [-f] [-l FILE] [-o FILE] [-v 0..5]
                                   [--version]
                                   referenceFile

Write reference sequence bases at SNP locations to a fasta file.
```

```
positional arguments:
  referenceFile         Relative or absolute path to the reference bases file
                        in fasta format

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           Force processing even when result file already exists
                        and is newer than inputs (default: False)
  -l FILE, --snpListFile FILE
                        Relative or absolute path to the SNP list file
                        (default: snplist.txt)
  -o FILE, --output FILE
                        Output file. Relative or absolute path to the SNP
                        reference sequence file (default: referenceSNP.fasta)
  -v 0..5, --verbose 0..5
                        Verbose message level (0=no info, 5=lots) (default: 1)
  --version             show program's version number and exit
```

collectSampleMetrics.sh

```
usage: collectSampleMetrics.sh [-h] [-f] [-c FILE] [-m INT ] [-o FILE] [-v FILE]␣
↪sampleDir referenceFile

Collect alignment, coverage, and variant metrics for a single specified sample.

Positional arguments:
  sampleDir          : Relative or absolute directory of the sample
  referenceFile      : Relative or absolute path to the reference fasta file

Options:
  -h                 : Show this help message and exit
  -f                 : Force processing even when result files already exist and
                       are newer than inputs
  -c FILE            : Relative or absolute path to the consensus fasta file
                       (default: consensus.fasta in the sampleDir)
  -C FILE            : Relative or absolute path to the consensus preserved fasta file
                       (default: consensus_preserved.fasta in the sampleDir)
  -m INT             : Maximum allowed number of SNPs per sample. (default: -1)
  -o FILE            : Output file. Relative or absolute path to the metrics file
                       (default: metrics in the sampleDir)
  -v FILE            : Relative or absolute path to the consensus vcf file
                       (default: consensus.vcf in the sampleDir)
  -V FILE            : Relative or absolute path to the consensus preserved vcf file
                       (default: consensus_preserved.vcf in the sampleDir)
```

combineSampleMetrics.sh

```
usage: combineSampleMetrics.sh [-h] [-n NAME] [-o FILE] sampleDirsFile

Combine the metrics from all samples into a single table of metrics for all samples.
The output is a tab-separated-values file with a row for each sample and a column
for each metric.
```

```
Before running this command, the metrics for each sample must be created by the
collectSampleMetrics.sh script.

Positional arguments:
  sampleDirsFile   : Relative or absolute path to file containing a list of
                     directories -- one per sample

Options:
  -h               : Show this help message and exit
  -n NAME          : File name of the metrics files which must exist in each of
                     the sample directories. (default: metrics)
  -o FILE          : Output file. Relative or absolute path to the combined metrics
                     file. (default: stdout)
  -s               : Emit column headings with spaces instead of underscores
```

CHAPTER 9

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/CFSAN-Biostatistics/snp-pipeline/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

SNP Pipeline could always use more documentation, whether as part of the official SNP Pipeline docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/CFSAN-Biostatistics/snp-pipeline/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

Get Started!

Ready to contribute? Here's how to set up *snp-pipeline* for local development.

1. Fork the *snp-pipeline* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/snp-pipeline.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv snppipeline
$ cd snppipeline/
$ python setup.py develop
$ pip install sphinx_rtd_theme     # the documentation uses the ReadTheDocs theme
```

4. Run the unit tests on the supplied data set to verify your installation is working:

```
$ python setup.py test
```

5. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

6. When you're done making changes, check that your changes pass the tests, including testing other Python versions:

```
$ python setup.py test
$ tox
- or -
$ . run_tests.sh -c # source this script to test other python versions without␣
↪using tox
```

To get tox, just pip install it into your virtualenv.

7. Run the regression tests:

```
$ test/regression_tests.sh
```

To get shunit2, install from https://code.google.com/p/shunit2/

8. Update the documentation and review the changes locally with sphinx:

```
$ cd docs
$ sphinx-build -b html . ./_build
$ xdg-open _build/index.html
```

9. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

10. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4, and 3.5, and for PyPI.

Tips

To run a subset of tests:

```
$ python -m unittest test.test_snppipeline
$ python -m unittest test.test_utils
```

CHAPTER 10

Credits

CFSAN BioInformatics Team

- Errol Strain
- Yan Luo
- James Pettengill
- Hugh A. Rand
- Steve Davis
- Justin Payne
- Al Shpuntoff
- Joseph D. Baugher
- Yu Wang

External Contributors

None yet. Why not be the first?

CHAPTER 11

History

0.7.0 (2016-11-30)

- Added a new script to the pipeline: `snp_filter.py` removes snps from the ends of contigs and from regions where the snp density is abnormally high. This is an important change to the pipeline with additional processing and new output files. See *SNP Filtering*.
- NOTE: You cannot re-use an old configuration file when running SNP Pipeline version 0.7.0. You must create a new configuration file. See *Configuration*.
- Fixed compatibility with bcftools 1.2 and higher.
- Updated the result files in the included data sets with the results obtained using bcftools v1.3.1 and bowtie2 v2.2.9. Note: upgrading from bowtie 2.2.2 to 2.2.9 did not change the snp matrix on any of the included datasets.

0.6.1 (2016-05-23)

- Fixed compatibility with SAMtools 1.3.
- Changed the expected results data sets to match the results obtained using SAMtools version 1.3.1. Starting with SAMtools version 1.0, the samtools mpileup command implemented a feature to avoid double counting the read depth when the two ends of a paired-end read overlap. If you use this feature of SAMtools, the pileup depth will be noticably reduced. You can still count the overlapping read sections twice by using SAMtools v0.1.19 or by using a configuration file specifying the `-x` option in `SamtoolsMpileup_ExtraParams`.
- Removed the obsolete `reads.snp.pileup` files from the included results data sets.

0.6.0 (2016-04-11)

Bug fixes:

- Fixed compatibility with the newly released PyVCF 0.6.8 package.

Other Changes:

- A new configuration parameter, `SnpPipeline_MaxSnps`, controls the maximum number of snps allowed for each sample. Samples with excessive snps exceeding this limit are excluded from the snp list and snp matrix. See *Excessive SNPs*.
- A new column in the metrics.tsv file, `Excluded_Sample`, indicates when a sample has been excluded from the snp matrix. This column is normally blank.
- Added a new script to the pipeline: `calculate_snp_distances.py` computes the SNP distances between all pairs of samples. The SNP distances are written to the output files `snp_distance_pairwise.tsv` and `snp_distance_matrix.tsv`.
- Changed Sun Grid Engine execution to use array-slot dependency where possible, resulting in less idle time waiting for job steps to complete.

0.5.2 (2016-03-07)

Bug fixes:

- An empty snplist.txt file should not cause errors when creating the referenceSNP.fasta.
- An empty snplist.txt file should not preclude re-running subsequent steps of the pipeline.
- When configured to ignore single-sample errors, a missing var.flt.vcf file should not preclude rebuilding the snplist.txt file during a pipeline re-run.
- The metrics file did not properly capture the total number of snps per sample. See below for the details.

Other Changes:

- Capture separate metrics counting phase 1 snps (varscan) and phase 2 snps (consensus). Previously, the metrics only included phase 1 snps. This changes the contents of both the `metrics` and `metrics.tsv` files. The metrics file now contains a new tag `phase1Snps`. The old tag `snps` now correctly counts the total number of snps. The metrics.tsv file now has separate column headers for phase 1 snps and phase 2 snps. Any code that parses those files may need modifications to work properly with v0.5.2.
- Added the `Average Insert Size` metric.
- The metrics.tsv column headings now contain underscores instead of spaces for better interoperability with some downstream analysis tools. Column headings with spaces can be generated by specifing the combineSampleMetrics.sh `-s` option in the configuration file.
- Remove the dependence on the snp matrix when collecting sample metrics.
- Improve the speed of metrics calculation when rerunning the pipeline. Reuse the previously computed metrics when recalculation would be slow.

0.5.1 (2016-02-19)

Bug fixes:

- Do not shutdown the pipeline when the generated snplist is empty when there are no snps.
- Do not attempt to merge VCF files when there are fewer than two VCF files to merge.

Other Changes:

- Added the `vcfFailedSnpGt` option to the call_consensus.py script to control how the VCF file GT data element is emitted when the snp is failed because of depth, allele frequency, or some other filter. If not specified, the GT element will contain a dot. Prior to this release, the behavior was to emit the ALT allele index. The old behavior can be retained by setting `--vcfFailedSnpGt 1`
- Changed the setup to require PyVCF version 0.6.7 or higher. It will automatically upgrade if necessary.
- Added error checking after running SamTools and VarScan to detect missing, empty, or erroneous output files.

0.5.0 (2016-01-19)

Bug fixes:

- Changed VCF file generator to not emit multiple alleles when the reference base is lowercase.

Other Changes:

- Trap errors, shutdown the pipeline, and prevent execution of subsequent steps when earlier processing steps fail. A summary of errors is written to the `error.log` file. See *Error Handling*.
- Check for the necessary software tools (bowtie, samtools, etc.) on the path at the start of each pipeline run.
- Check for missing or empty input files at the start of each processing step.
- Added two new parameters, `GridEngine_QsubExtraParams` and `Torque_QsubExtraParams`, to the configuration file to pass options to qsub when running the SNP Pipeline on an HPC computing cluster. Among other things, you can control which queue the snp-pipeline will use when executing on an HPC with multiple queues. See *Configuration*.
- Removed the "job." prefix to shorten job names when running on an HPC.
- Changed the vcf file generator to emit reference bases in uppercase. Added the `vcfPreserveRefCase` flag to the call_consensus.py script to cause the vcf file generator to emit each reference base in uppercase/lowercase as it appears in the original reference sequence file. If not specified, the reference bases are emitted in uppercase. Prior to this release, the behavior was to always preserve the original case.
- Added support for Python 3.3, 3.4, 3.5.
- Implemented a regression test suite for the bash shell scripts, using the shUnit2 package.

0.4.1 (2015-10-30)

Bug fixes:

- Fixed a Python 2.6 incompatibility with the new consensus caller.

Other Changes:

- Added Tox support for automatically testing installation and execution with multiple Python versions.

0.4.0 (2015-10-22)

Bug fixes:

- When run on Grid Engine with the default settings, bowtie2 was consuming all available CPU cores per node while scheduled with Grid to use only 8 cores. On a lightly loaded cluster, this bug made the pipeline run faster, but when the cluster was full or nearly full, it would cause contention for available CPU resources and cause jobs to run more slowly. Changed to use only 8 CPU cores by default.
- The consensus snp caller miscounted the number of reference bases when the pileup record contained the ^ symbol marking the start of a read segment followed by a dot or comma. In this situation, the dot or comma should not be counted as reference bases.

Other Changes:

- Added support for the Smalt aligner. You can choose either bowtie2 or smalt in the configuration file. A new parameter in the configuration file, `SnpPipeline_Aligner`, selects the aligner to use. Two additional configuration parameters, `SmaltIndex_ExtraParams` and `SmaltAlign_ExtraParams` can be configured with any Smalt command line options. See *Tool Selection*. The default aligner is still bowtie2.
- Split the create_snp_matrix.py script into two pieces. The new script, call_consensus.py, is a redesigned consensus caller which is run in parallel to call snps for multiple samples concurrently. The create_snp_matrix.py script simply merges the consensus calls for all samples into a multi-fasta file.
- The new consensus caller has the following adjustable parameters. See the *call_consensus.py* command reference.
  - `minBaseQual` : Mimimum base quality score to count a read.
  - `minConsFreq` : Minimum consensus frequency.
  - `minConsStrdDpth` : Minimum consensus-supporting strand depth.
  - `minConsStrdBias`: Strand bias.
- Added the capability to generate VCF files. By default, a file named consensus.vcf is generated by the consensus caller for each sample, and the merged multi-sample VCF file is called snpma.vcf. This capability introduces a new dependency on bgzip, tabix, and bcftools. You can disable VCF file generation by removing the `--vcfFileName` option in the configuration file. Also, be aware the contents of the VCF files may change in future versions of the SNP Pipeline.
- Added configuration parameters `Torque_StripJobArraySuffix` and `GridEngine_StripJobArraySuffix` to improve compatibility with some HPC environments where array job id suffix stripping is incompatible with qsub.
- Renamed the configuration parameter `PEname` to `GridEngine_PEname`.

0.3.4 (2015-06-25)

Bug fixes:

- The referenceSNP.fasta file was missing newlines between sequences when the reference fasta file contained multiple sequences. In addition, each sequence was written as a single long string of characters. Changed to emit a valid fasta file. Updated the expected result files for the datasets included with the distribution accordingly.
- Changed the run_snp_pipeline.sh script to allow blank lines in the file of sample directories when called with the -S option.
- Changed the run_snp_pipeline.sh script to allow trailing slashes in the file of sample directories when called with the -S option.
- Do not print system environment information when the user only requests command line help.
- Fixed the broken pypi downloads per month badge on the readme page.

Other Changes:

- Changed the default configuration file to specify the `-X 1000` option to the bowtie2 aligner. This parameter is the maximum inter-mate distance (as measured from the furthest extremes of the mates) for valid concordant paired-end alignments. Previously this value was not explicitly set and defaulted to 500. As a result of this change, the generated SAM files may have a different number of mapped reads, the pileup files may have different depth, and the number of snps called may change.
- We now recommend using VarScan version 2.3.9 or later. We discovered VarScan v2.3.6 was occasionally omitting the header section of the generated VCF files. This in turn, caused the SNP Pipeline to miss the first snp in the VCF file. This is not a SNP Pipeline code change, only a documentation and procedural change.
- Updated the result files in the included data sets with the results obtained using VarScan v2.3.9 and the Bowtie -X 1000 option.
- Log the Java classpath to help determine which version of VarScan is executed.
- Changed the python unit tests to execute the non-python processes in a temporary directory instead of assuming the processes were already run in the test directory.

0.3.3 (2015-04-14)

Bug fixes:

- Improve HPC qsub submission speed throttling to avoid errors with the HPC job scheduler when submitting large and small jobs. Dynamically adjust the delays between HPC array job submission so small datasets have small delays and large datasets have large delays between qsub submissions.
- Process the sample directories in order by size, largest first, considering only the size of fastq files and ignoring all other files. Previously non-fastq files were affecting the processing order.
- Fixed divide-by-zero error in create_snp_matrix when no snps are detected.
- Don't skip the last sample when run_snp_pipeline is started with the -S option and the file of sample directories is not terminated with a newline.
- Gracefully exit run_snp_pipeline with error messages when run with -S option and any of the sample directories in the sample directory file is missing, empty, or does not contain fastq files.
- Gracefully exit run_snp_pipeline with an error message when run with -s option and the samples directory is empty or contains no subdirectories with fastq files.
- Fixed the sun grid engine "undefined" task id reported in non-array job log files.

Other Changes:

- Sample Metrics. The pipeline generates a table of sample metrics capturing various alignment, coverage, and snp statistics per sample. See *Metrics*.
- Explicitly expose the `minConsFreq` parameter in the supplied default configuration file to make it easier to adjust.
- Updated the FAQ with instructions to install to an older version.

0.3.2 (2015-01-14)

Bug fixes:

- Fixed (again) a Python 2.6 incompatibility with formatting syntax when printing the available RAM. This affected the shell scripts (prepReference.sh, alignSampleToReference.sh, prepSamples.sh).
- Improved installation in a Python 2.6 environment. Added several Python packages to the automatic setup script.

Other Changes:

- Added support for the Grid Engine job queue manager. See *High Performance Computing*.
- Added a configurable parameter, `minConsFreq`, to the create_snp_matrix.py script. This parameter specifies the mimimum fraction of reads that must agree at a position to make a consensus call. Prior to version 0.3.2, the snp pipeline required that a majority (more than half) of the reads must agree to make a snp call. In version 0.3.2, the default behavior requires at least 60% of reads must agree to make a consensus call.
- Changed the included snp matrix files for the agona and listeria data sets to match the new results obtained by setting minConsFreq=0.6. The lambda virus results were not impacted by this change.
- Revised the Installation instructions with more detailed step-by-step procedures.
- Added a Dockerfile for automated docker builds. This feature is still experimental.

0.3.1 (2014-10-27)

Bug fixes:

- Fixed a Python 2.6 incompatibility with formatting syntax when printing the available RAM. Also added the Python version to the log files.

0.3.0 (2014-10-22)

Bug fixes:

- Fixed some Mac OSX incompatibilities.
- Fixed a bug in copy_snppipeline_data.py that caused copy failure when the destination directory did not exist.
- Fixed alignSampleToReference.sh to properly handle unpaired gzipped fastq files.

Installation Changes:

- There is a new dependency on the python psutil package. When you install the SNP Pipeline, pip will attempt to install the psutil package automatically. If it fails, you may need to manually install the python-dev package. In Ubuntu, `sudo apt-get install python-dev`

Other Changes:

Note a possible loss of backward compatibilty for existing workflows using alignSampleToReference.sh and prepSamples.sh

- All-in-one script: Added a new script, run_snp_pipeline.sh, to run the entire pipeline either on a workstation or on a High Performance Computing cluster with the Torque job queue manager. See *All-In-One SNP Pipeline Script*.
- Logging: The run_snp_pipeline.sh script adds consistent logging functionality for workstation and HPC runs. The logs for each pipeline run are stored in a time-stamped directory under the output directory. See *Logging*.
- Timestamp checking: Changed the python scripts (create_snp_list.py, create_snp_pileup.py, create_snp_matrix.py, create_snp_reference.py) to skip processing steps when result files already exist and are newer than the input files. If you modify an upstream file, any dependent downstream files will be rebuilt. You

can force processing regardless of file timestamps with the `-f` option. Similar functionality for the shell scripts was previously implemented in release 0.2.0.

- Mirrored input files: The run_snp_pipeline.sh script has the capability to make a mirrored copy of the input reference and samples to avoid polluting a clean repository. You have the choice to create copies, soft links, or hard links. See *Mirrored Inputs*.
- Configuration file: Added the capability to customize the behavior of the SNP Pipeline by specifying parameters either in a configuration file, or in environment variables. You can create a configuration file with default values pre-set by executing `copy_snppipeline_data.py configurationFile` from the command line. Pass the configuration file to the run_snp_pipeline.sh script with the `-c` option. Alternatively, environment variables matching the names of the parameters in the configuration file can be manually set (be sure to export the variables). When the run_snp_pipeline.sh script is run, it copies the configuration file for the run into the log directory for the run. See *Configuration*.
- Removed the `-p INT` command line option, to specify the number of cpu cores, from the alignSampleToReference.sh script. You can now control the number of cpu cores used by bowtie2 with the `-p INT` option either in the configuration file when running run_snp_pipeline.sh, or in the `Bowtie2Align_ExtraParams` environment variable when running alignSampleToReference.sh directly. If not specified, it defaults to 8 cpu cores on a HPC cluster, or all cpu cores on a workstation.
- Removed the `--min-var-freq 0.90` varscan mpileup2snp option from the prepSamples.sh script. This parameter is now specified in the `VarscanMpileup2snp_ExtraParams` environment variable or in the configuration file.
- Listeria monocytogenes data set: Added a Listeria monocytogenes data set. Updated the usage instructions, illustrating how to download the Listeria samples from NCBI and how to run the SNP Pipeline on the Listeria data set. The distribution includes the expected result files for the Listeria data set. Note that due to the large file sizes, the Listeria expected results data set does not contain all the intermediate output files.
- Added a command reference page to the documentation. See *Command Reference*.

0.2.1 (2014-09-24)

Bug fixes:

- Version 0.2.0 was missing the Agona data files in the Python distribution. The GitHub repo was fine. The missing files only impacted PyPi. Add the Agona data files to the Python distribution file list.

0.2.0 (2014-09-17)

Changes Impacting Results:

- Previously, the pipeline executed SAMtools mpileup twice – the first pileup across the whole genome, and the second pileup restricted to those positions where snps were identified by varscan in *any* of the samples. This release removes the second SAMtools pileup, and generates the snp pileup file by simply extracting a subset of the pileup records from the genome-wide pileup at the positions where variants were found in *any* sample. The consequence of this change is faster run times, but also an improvement to the results – there will be fewer missing values in the snp matrix.
- Changed the the supplied lambda virus expected results data set to match the results obtained with the pipeline enhancements in this release and now using SAMtools version 0.1.19. SAMtools mpileup version 0.1.19 excludes read bases with low quality. As a reminder, the expected results files are fetched with the copy_snppipeline_data.py script.
- Removed the “<unknown description>” from the snp matrix fasta file.

Other Changes:

Note the loss of backward compatibilty for existing workflows using prepReference.sh, alignSampleToReference.sh, prepSamples.sh, create_snp_matrix.py

- Split the create_snp_matrix script into 4 smaller scripts to simplify the code and improve performance when processing many samples in parallel. Refer to the *Usage* section for the revised step-by-step usage instructions. The rewritten python scripts emit their version number, arguments, run timestamps, and other diagnostic information to stdout.
- Changed the default name of the reads.pileup file to reads.snp.pileup. You can override this on the command line of the create_snp_pileup.py script.
- Added the referenceSNP.fasta file to the supplied lambda virus expected results data set.
- Updated the usage instructions, illustrating how to download the Agona samples from NCBI and how to run the SNP Pipeline on the Agona data set.
- Updated the supplied expected result files for the Agona data set. Note that due to the large file sizes, the Agona expected results data set does not contain all the intermediate output files.
- Improved the online help (usage) for all scripts.
- The copy_snppipeline_data.py script handles existing destination directories more sensibly now. The example data is copied into the destination directory if the directory already exists. Otherwise the destination directory is created and the example data files are copied there.
- Changed the alignSampleToReference.sh script to specify the number of CPU cores with the -p flag, rather than a positional argument. By default, all CPU cores are utilized during the alignment.
- Changed the shell scripts (prepReference.sh, alignSampleToReference.sh, prepSamples.sh) to expect the full file name of the reference including the fasta extension, if any.
- Changed the shell scripts (prepReference.sh, alignSampleToReference.sh, prepSamples.sh) to skip processing steps when result files already exist and are newer than the input files. If you modify an upstream file, any dependent downstream files will be rebuilt. You can force processing regardless of file timestamps with the `-f` option.
- Changed the name of the sorted bam file to reads.sorted.bam.
- Changed the general-case usage instructions to handle a variety of fastq file extensions (*.fastq* and *.fq*).

0.1.1 (2014-07-28)

Bug fixes:

- The snp list, snp matrix, and referenceSNP files were incorrectly sorted by position alphabetically, not numerically.
- The SNP Pipeline produced slightly different pileups each time we ran the pipeline. Often we noticed two adjacent read-bases swapped in the pileup files. This was caused by utilizing multiple CPU cores during the bowtie alignment. The output records in the SAM file were written in non-deterministic order when bowtie ran with multiple concurrent threads. Fixed by adding the `--reorder` option to the bowtie alignment command line.
- The snp list was written to the wrong file path when the main working directory was not specified with a trailing slash.

Other Changes:

Note the loss of backward compatibilty for existing workflows using prepSamples.sh

- Moved the bowtie alignment to a new script, alignSampleToReference.sh, for better control of CPU core utilization when running in HPC environment.
- Changed the prepSamples.sh calling convention to take the sample directory, not the sample files.
- prepSamples.sh uses the CLASSPATH environment variable to locate VarScan.jar.
- Changed prepReference.sh to run `samtools faidx` on the reference. This prevents errors later when multiple samtools mpileup processes run concurrently. When the faidx file does not already exist, multiple samtools mpileup processes could interfere with each other by attempting to create it at the same time.
- Added the intermediate lambda virus result files (*.sam, *.pileup, *.vcf) to the distribution to help test the installation and functionality.
- Changed the usage instructions to make use of all CPU cores.
- Log the executed commands (bowtie, samtools, varscan) with all options to stdout.

0.1.0 (2014-07-03)

- Basic functionality implemented.
- Lambda virus tests created and pass.
- 19. Agona tests created – UNDER DEVELOPMENT
- Installs properly from PyPI.
- Documentation available at ReadTheDocs.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search