Phoenix Documentation

Release 1.0

Public Health England

04 September, 2018

Contents

This documentation is designed to give an overview as well as detailed API reference for Public Health England's single nucleotide polymorphism calling pipeline. Currently the pipeline does not provide stand alone means of calling SNPs, instead it interfaces with other published tool e.g. BWA and samtools.

Contents:

CHAPTER 1

Introduction

Author Public Health England

Date 04 September, 2018

1.1 Installation

From source:

```
git clone https://github.com/phe-bioinformatics/PHEnix.git
pip2 install -e PHEnix
```

Directly from github:

```
pip install git+https://github.com/phe-bioinformatics/PHEnix.git
```

Note: Installing from Pip - Coming Soon.

1.2 Overview

This code was designed to allow users to input fastq files and a reference sequence and perform:

- Reference mapping
- VCF generation
- VCF filtering
- FASTA sequence of SNPs

The process is comprised of three steps:

- Reference sequence preparation (prepare_reference.py)
- Mapping and filtered VCF generation (run_snp_pipeline.py)
- FASTA generation from single or multiple VCFs (vcf2fasta.py)

Example:

prepare a bwa and gatk reference using a fasta file (myref.fasta)

```
phenix.py prepare_reference \
--mapper bwa \
--variant gatk \
--reference myref.fasta
```

map, call and filter variants on fastq files (my.R1.fastq, my.R2.fastq). Filter SNPs on minimum depth, mapping quality and AD ratio

```
phenix.py run_snp_pipeline \
-r1 my.R1.fastq \
-r2 my.R2.fastq \
-r myref.fasta \
--sample-name mysample \
--mapper bwa \
--variant gatk \
--filters min_depth:5,mq_score:30,ad_ratio:0.9
```

generate a FASTA file of SNPs using filtered VCFs in current directory

```
phenix.py vcf2fastq -d ./ -o output.fasta --regex filtered
```

1.3 Requirements

A lot of functionality depends on the presence of existing 3rd party tools:

Mappers:

- BWA - Download from [https://github.com/lh3/bwa]
- Bowtie2 - Download from [https://github.com/BenLangmead/bowtie2]

Variant Caller:

- GATK - Download from [https://www.broadinstitute.org/gatk/download/] * Picard - Download from [http://broadinstitute.github.io/picard/]
- MPileup - Download from [https://github.com/samtools/samtools]

In order for them to function properly, they need to be already in you **PATH**. For commands that run through Java archives, please set appropriate environment variable (see below).

1.4 Python

- Python >= 2.7
- argparse
- PyVCF

- PyYAML
- matplotlib (_optional_)
- bintrees (_optional_) - If you are using **vcf2fasta.py**
- numpy (_optional_)
- matplotlib.venn (_optional_) - psycopg2 (_optional_)

1.5 3rd Party Requirements

1.6 Samtools

Samtools Samtools can be downloaded from https://github.com/samtools/samtools. It is used to filter and convert to SAM/BAM files and in mpileup variant caller.

1.7 BCFTools

BCFtools can be downloaded from https://github.com/samtools/bcftools. It is used for calling variants in mpileup.

BWA Heng Li's mapper can be downloaded from https://github.com/lh3/bwa.

Bowtie2 Bowtie2 mapper available from https://github.com/BenLangmead/bowtie2.

GATK Set *GATK_JAR* - full path to the GATK Java archive.

1.8 Picard Tools

Picard is needed for GATK to create dictionary of reference fasta. Either set *PICARD_TOOLS_PATH* - path to directory where different Picard jars are or set *PICARD_JAR* - path to **picard.jar**. Older Picard distributions have many different jars (use first suggestion), where as newer versions have merged all into one jar file.

CHAPTER 2

Prerequisites

The refence needs to be index in an appropriate way for different mapper/variant callers. This can be done manually for specific tools or using:

```
prepare_reference.py --mapper [bwa | bowtie2] \
--variant [gatk | mpileup] \
--reference <path_to_reference>
```

Internally, for the pipeline, the indexing will be done automatically, because software assumes that a reference is present for each sample. It has to be done differently for other use cases because if multiple processes trying to index the same file, this may lead to corrupted indeces.

Note: The reference file can't have *fas* extension, because Picard Tools throws an exception.

CHAPTER 3

Filters

One of the key parts of the VCF processing is to filter quality calls. To do this we have created a flexible interface with variaety of filters avaialble:

- **qual_score** - Filter records where **QUAL** score is below given threshold.
- **ad_ratio** - Filter, defined by **gatk**, records where ratio of alt allele to sum of all alleles is below given fraction.
- **dp4_ratio** - Similar to **ad_ratio**, but used in **mpileup** variant caller.
- **mq_score**- Filter records that fall below specified **MQ** score (from _INFO_ field).
- **mq0_ratio** - Filter, defined by **gatk**, records with **MQ0** to **DP** ratio _above_ given threshold (both values are from _INFO_ field).
- **mq0f_ratio** - Similar to the **mq0_ratio**, but used in **mpileup** variant caller.
- **qg_score** - Filter records that fall below specified **GQ** score (from **first** sample).
- **min_depth** - Filter records with mean depth below specified threshold (**DP** from sample field).
- **uncall_gt** - Filter records with uncallable genotypes in VCF (**GT** is ./.).

All filters are applied for each position and those positions that pass ALL filters are kept as quality calls. Positions failing filter will be kept for future reference and creating fasta files, when needed. To specify filters to be used, simply list them as key:threshold pairs separated by comma(,). For filters that don't require threshold, leave blank after ':'.

Not all filters are available for all variant callers. Which filters can be used with your data depends on the variant caller that was used to create your VCF file.

Filter	Remark
Quality score	GATK and samtools mpileup
AD ratio	GATK and samtools mpileup version >=1.3
DP4 ratio	samtools mpileup version <=1.2
MQ score	GATK and samtools mpileup
MQ0 ratio	GATK only
MQ0F ratio	samtools mpileup only
GQ score	GATK and samtools mpileup
Minimum depth	GATK and samtools mpileup
Uncall GT	GATK and samtools mpileup

If you want to filter a VCF file that was not created with either GATK or samtools, please refer to the documentation of this tool to check which data is available in your VCF files.

CHAPTER 4

Annotators

Individual VCFs can be annotated using custom annotators. Currently available annotators:

- **coverage** - Annotates with information about _mean_ and _dev_ of the depth in the VCF (using DP from INFO).

The data can be accessed from the metadata field in the VCF in the following way:

```
r = vcf.Reader(filename="/path/to/vcf")
print r.metadata["coverageMetaData"][0]["mean"]
print r.metadata["coverageMetaData"][0]["dev"]
```

CHAPTER 5

Calling SNPs

If you have a sample and you want to have one-stop-shop analysis run the following:

```
$ phenix.py run_snp_pipeline \
-r1 <path to R1.fastq> \
-r2 <path to R2.fastq> \
-r <path to reference> \
--sample-name <NAME> \
--mapper bwa \
--variant gatk \
--filters min_depth:5,mq_score:30 \
-o <path to output directory>
```

This will map with **BWA** and call variants with **GATK**. Intermediate files are written into the specified **output directory**. **–sample-name** option is very important, it specifies what output files will be called and the read group in the BAM file. If you omit it, **test_sample** will be used.

5.1 Config

If you have a pipeline, or want to run the same settings for different samples this process can be simplefied by having a single config file. The file is in **YAML** format and takes common options that may be used for all samples:

- mapper - **string**
- mapper-options - **string**
- variant - **string**
- variant-options - **string**
- filters - **dictionary**
- annotators - **list**

Below is an example of config file:

```
mapper: bwa
mapper-options: -t 4
variant: gatk
variant-options: --sample_ploidy 2 --genotype_likelihoods_model SNP -rf BadCigar -out_
↪mode EMIT_ALL_SITES -nt 1
filters:
  ad_ratio: 0.9
  min_depth: 5
  qual_score: 30
  mq_score: 30
annotators:
  - coverage
```

CHAPTER 6

Converting to FASTA

A lot of downstream applications take on FASTA formated files, not VCFs. We have included a script for converting VCF data to FASTA format.

```
$ vcfs2fasta -d <path to directory of VCFs> -o <path to output FASTA>
```

This tool is also able to filter out samples and columns that may be bad quality. E.g. **–sample-Ns** specifies maximum fraction of Ns present in a sample. **–Ns** specifies maximum fraction of Ns allowed per column. **–with-mixtures** can specify if mixed position should be output as over certain fraction. First, samples are sifted out then columns are checked. **–with-stats** allows to output genomic positions of the called SNPs. Each line coresponds to a character position in the FASTA file. E.g. to get the position of the 23rd snp, go to 23rd line in the positions file. With this option, if numpy and matplotlib are installed, **plots** directory will be created and a summary plot is generated, summarising called SNPs, Ns, mixtures, and bases with 0 depth. Providing **–with-dist-matrix** will also write distance matrix to the specified path.

CHAPTER 7

API Reference

7.1 Filters

One of the key aims of this project is to provide a flexible and extendable interface for filtering VCFs. While there are a number of tools that will process a VCF file, we felt that none offered in depth understanding that we wanted. Instead we have based this library on PyVCF and extended with variety of filters *Filters*.

If you feel that there is a filter that has not been implemented, or you want to try your hand at implementing object oriented classes, then please see *Implementing Filter*.

7.2 Mappers

We found there was no single neat way of calling different mappers (there is BioPython, of course) but it wasn't feasible to integrate it into our software stack in the way that we wanted to. Instead, we wrote a simple interfaces that can be easily extended to add new mappers. See *Implementing Mapper*.

7.3 Variant Callers

Just like with mappers, there was no neat way of incorporating other interfaces into this software stack. To add your favourite variant caller see *Implementing Variant Caller*.

7.3.1 phe

Subpackages

phe.annotations

Submodules

phe.annotations.CoverageAnnotator module

Date 10 Nov, 2015

Author Alex Jironkin

class CoverageAnnotator(*config=None*)
Bases: *phe.annotations.Annotator*

classdocs

Methods

annotate	
get_meta	
get_meta_values	

annotate(*vcf_path=None*)

get_meta_values()

name = 'coverage'

Module contents

Date 10 Nov, 2015

Author Alex Jironkin

class Annotator(*name*)
Bases: *phe.metadata.PHEMetaData*

Base class for Annotators

Attributes

name

Methods

annotate	
get_meta	
get_meta_values	

annotate(*vcf_path=None*)

get_meta()
Get the metadata.

get_meta_values()

name = None

available_annotators()
Return list of available filters.

dynamic_annotator_loader()
Fancy way of dynamically importing existing annotators.

Returns

dict: Available annotators dictionary. Keys are parameters that can be supplied to the filters.

make_annotators(*config*)
Create a list of annotators from *config*.

Parameters

config: dict, optional Dictionary with name: value pairs. For each name, an appropriate Annotator will be found and instanciated.

Returns

list: List of *Annotator* annotators.

phe.mapping

Submodules

phe.mapping.BWAMapper module

Implementation of the Mapper class using BWA (Heng Li) mapper.

Date 17 Sep, 2015

Author Alex Jironkin

class BWAMapper(*cmd_options=None*)
Bases: *phe.mapping.Mapper*

BWA mapper developed by Heng Li.

Methods

get_samtools_version()	Get version of samtools used.
make_bam(*args, **kwargs)	Make **indexed** BAM from reference, and fastq files.
validate()	Validate itself, by checking appropriate commands can run.

create_aux_files	
get_info	
get_meta	
get_version	
make_sam	

create_aux_files(*ref*)
Create required auxiliary files for *ref*.

Parameters

ref: str Path to the reference file.

Returns

bool: True if auxiliary files were created, False otherwise.

get_info(*plain=False*)
Get information about the mapper.

get_version()
Get the version of the underlying command used.

make_sam(**args*, ***kwargs*)
Make SAM from reference, and fastq files.

Parameters

ref: str Path to the reference file used for mapping.

R1: str Path to the R1/Forward reads file in FastQ format.

R2: str: Path to the R2/Reverse reads file in FastQ format.

out_file: str Name of the output file where SAM data is written.

sample_name: str Name of the sample to be included in the read group (RG) field.

make_aux: bool, optional True/False to make auxilliary files (default: False).

Returns

bool: True iff mapping returns 0, False otherwise.

name = 'bwa'
Plain text name of the mapper.

phe.mapping.Bowtie2Mapper module

Implementation of the Mapper class using Bowtie2 mapper.

Date 17 Sep, 2015

Author Alex Jironkin

class Bowtie2Mapper(*cmd_options=None*)
Bases: *phe.mapping.Mapper*

Bowtie2 mapper

Methods

get_samtools_version()	Get version of samtools used.
make_bam(*args, **kwargs)	Make **indexed** BAM from reference, and fastq files.
validate()	Validate itself, by checking appropriate commands can run.

create_aux_files	
get_info	
get_meta	
get_version	
make_sam	

create_aux_files(*ref*)
Create required auxiliary files for *ref*.

Parameters

ref: str Path to the reference file.

Returns

bool: True if auxiliary files were created, False otherwise.

get_info(*plain=False*)
Get information about the mapper.

get_version()
Get the version of the underlying command used.

make_sam(**args*, ***kwargs*)
Make SAM from reference, and fastq files.

Parameters

ref: str Path to the reference file used for mapping.

R1: str Path to the R1/Forward reads file in FastQ format.

R2: str: Path to the R2/Reverse reads file in FastQ format.

out_file: str Name of the output file where SAM data is written.

sample_name: str Name of the sample to be included in the read group (RG) field.

make_aux: bool, optional True/False to make auxilliary files (default: False).

Returns

bool: True iff mapping returns 0, False otherwise.

name = 'bowtie2'
Plain text name of the mapper.

phe.mapping.mapping_factory module

Date 22 Sep, 2015

Author Alex Jironkin

available_mappers()
Return a list of available mappers.

dynamic_mapper_loader()
Fancy way of dynamically importing existing mappers.

Returns

dict: Available mappers dictionary. Keys are names that can be used for existing mapper implementations.

factory(*mapper=None*, *custom_options=None*)
Create an instance of a mapper from _config_.

Parameters

mapper: str, optional Name of the mapper to initialise.

custom_options: str, optional Custom options to be passed to the mapper.

Returns

:py:class:'phe.mapping.Mapper': Instance of a *phe.mapping.Mapper* with provided name, or None if the mapper could not be found.

Module contents

Mapping related classes and functions.

Implementing Mapper

This project does not provide novel way of mapping or calling variants. Instead it interfaces with available mappers through command line interface. Hopefully, we have made it straight forward to add your favourite mapper for others to use.

To do this, you will need to implement *Mapper* class. There are a number of methods that need to be implemented for you mapper to work:

- *Mapper.name* - Attribute that specifies the name of the mapper (used to dynamically load it).
- *Mapper.make_sam()* - Create a sam file in *out_file* from *R1/Forward*, *R2/Reverse* and *ref*. See *Mapper.make_sam()* for all input options.
- *Mapper.create_aux_files()* - Create auxilliary files needed for the mapper. This is also called by *phenix*.
- *Mapper.get_version()* - get the version of the mapper.
- *Mapper.get_info()* - Get the meta data information for the mapper (included in the VCF header).
- *Mapper.make_bam()* - If you want, you can also implement this method, but my default samtools is used to convert from SAM -> BAM.

Bear in mind that **cmd_options** will be passed on, if they specified in the command line or config under **mapper-options**.

Once you have implemented this interface, save the file in the *mapping* directory and it should be automatically picked up. To verify run:

```
run_snp_pipeline.py --help
```

You should see your mapper in the list of available mappers. If the mapper works, it will also be included n automatically in the documentations.

Date 22 Sep, 2015

Author Alex Jironkin

class Mapper(*cmd_options=None*)

Bases: *phe.metadata.PHEMetaData*

Abstract Mapper class that provides generic interface to the mapping implementation for a particular mapper.

Attributes

name Return name of the mapper (implemented by individual classes.

Methods

create_aux_files(ref)	Create required auxiliary files for *ref*.
get_info([plain])	Get information about the mapper.
get_samtools_version()	Get version of samtools used.
get_version()	Get the version of the underlying command used.
make_bam(*args, **kwargs)	Make **indexed** BAM from reference, and fastq files.
make_sam(*args, **kwargs)	Make SAM from reference, and fastq files.
validate()	Validate itself, by checking appropriate commands can run.

get_meta	

create_aux_files(*ref*)

Create required auxiliary files for *ref*.

Parameters

ref: str Path to the reference file.

Returns

bool: True if auxiliary files were created, False otherwise.

get_info(*plain=False*)

Get information about the mapper.

get_meta()

Get the metadata.

get_samtools_version()

Get version of samtools used. Reterned as a triple (major, minor, patch)

get_version()

Get the version of the underlying command used.

make_bam(**args*, ***kwargs*)

Make **indexed** BAM from reference, and fastq files.

Parameters

ref: str Path to the reference file used for mapping.

R1: str Path to the R1/Forward reads file in FastQ format.

R2: str: Path to the R2/Reverse reads file in FastQ format.

out_file: str Name of the output file where BAM data is written.

sample_name: str Name of the sample to be included in the read group (RG) field.

Returns

bool: True iff mapping, converting to BAM and indexing returns 0, False otherwise.

make_sam(**args*, ***kwargs*)
Make SAM from reference, and fastq files.

Parameters

ref: str Path to the reference file used for mapping.

R1: str Path to the R1/Forward reads file in FastQ format.

R2: str: Path to the R2/Reverse reads file in FastQ format.

out_file: str Name of the output file where SAM data is written.

sample_name: str Name of the sample to be included in the read group (RG) field.

make_aux: bool, optional True/False to make auxilliary files (default: False).

Returns

bool: True iff mapping returns 0, False otherwise.

name
Return name of the mapper (implemented by individual classes.

validate()
Validate itself, by checking appropriate commands can run.

phe.metadata

Module contents

Metadata related information.

Date 22 Sep, 2015

Author Alex Jironkin

class PHEMetaData
Bases: object

Abstract class to provide interface for meta-data creation.

Methods

get_meta()	Get the metadata.

get_meta()
Get the metadata.

phe.variant

Submodules

phe.variant.GATKVariantCaller module

Date 22 Sep, 2015

Author Alex Jironkin

class GATKVariantCaller(*cmd_options=None*)
Bases: *phe.variant.VariantCaller*

Implemetation of the Broad institute's variant caller.

Methods

create_aux_files(ref)	Create auxiliary files needed for this variant.
get_meta()	Get the metadata about this variant caller.

get_info	
get_version	
make_vcf	
validate	

create_aux_files(*ref*)
Create auxiliary files needed for this variant.

Tools needed: samtools and picard tools. Picard tools is a Java library that can be defined using environment variable: PICARD_JAR specifying path to picard.jar or PICARD_TOOLS_PATH specifying path to the directory where separate jars are (older version before jars were merged into a single picard.jar).

Parameters

ref: str Path to the reference file.

Returns

bool: True if auxiliary files were created, False otherwise.

get_info(*plain=False*)
Get information about this variant caller.

get_version()
Get the version of the underlying command used.

make_vcf(**args*, ***kwargs*)
Create a VCF from **BAM** file.

Parameters

ref: str Path to the reference file.

bam: str Path to the indexed **BAM** file for calling _variants.

vcf_file: str path to the VCF file where data will be written to.

make_aux: bool, optional True/False create auxilliary files (default: False).

Returns

bool: True if variant calling was successful, False otherwise.

name = 'gatk'
Plain text name of the variant caller.

validate()
Check if the command can run.

phe.variant.MPileupVariantCaller module

Date 22 Sep, 2015

Author Alex Jironkin

class MPileupVariantCaller(*cmd_options=None*)
Bases: *phe.variant.VariantCaller*

Implemetation of the Broad institute's variant caller.

Methods

create_aux_files(ref)	Index reference with faidx from samtools.
get_meta()	Get the metadata about this variant caller.
validate()	Check if the command can run.

get_info	
get_version	
make_vcf	

create_aux_files(*ref*)
Index reference with faidx from samtools.

Parameters

ref: str Path to the reference file.

Returns

bool: True if auxiliary files were created, False otherwise.

get_info(*plain=False*)
Get information about this variant caller.

get_version()
Get the version of the underlying command used.

make_vcf(**args*, ***kwargs*)
Create a VCF from **BAM** file.

Parameters

ref: str Path to the reference file.

bam: str Path to the indexed **BAM** file for calling _variants.

vcf_file: str path to the VCF file where data will be written to.

make_aux: bool, optional True/False create auxilliary files (default: False).

Returns

bool: True if variant calling was successful, False otherwise.

name = 'mpileup'
Plain text name of the variant caller.

phe.variant.variant_factory

Classes and functions for working with variant callers.

Date 22 Sep, 2015

Author Alex Jironkin

available_callers()
Return list of available variant callers.

dynamic_caller_loader()
Fancy way of dynamically importing existing variant callers.

Returns

dict: Available variant callers dictionary. Keys are parameters that can be used to call variants.

factory(*variant=None*, *custom_options=None*)
Make an instance of a variant class.

Parameters

variant: str, optional Name of the variant class to instantiate.

custom_options: str, optional Custom options to be passed directly to the implementing class.

Returns

:py:class:'phe.variant.VariantCaller': Instance of the *phe.variant.VariantCaller* for given variant name, or None if one couldn't be found.

Module contents

Classes and methods to work with _variants and such.

Implementing Variant Caller

Similarly to *Implementing Mapper* and *Implementing Filter* adding your variant caller is easy and sraight forward. Implement *VariantCaller* class with the following attributes/methods:

- `VariantCaller.name` - Name of your variant caller, used to dynamically load it.
- *VariantCaller.make_vcf()* - Create a VCF from input BAM. See *VariantCaller.make_vcf()* for input list.
- *VariantCaller.get_version()* - Return the version of this variant caller.
- *VariantCaller.get_info()* - Return information about this caller and included in the VCF header.
- *VariantCaller.create_aux_files()* - Create auxilliary files needed by the caller. Also used in *phenix*.

Save your new variant caller in the *variant* directiry and it will be automatically picked up by the system. To verify run:

```
run_snp_pipeline.py --help
```

It should appear in the list of available callers.

Date 22 Sep, 2015

Modified 04 September, 2018

Author Alex Jironkin

class VCFTemplate(*vcf_reader*)
Bases: `object`

This is a small hack class for the Template used in generating VCF file.

class VariantCaller(*cmd_options=None*)
Bases: *phe.metadata.PHEMetaData*

Abstract class used for access to the implemented variant callers.

Methods

create_aux_files(ref)	Create needed (if any) auxiliary files.
get_info([plain])	Get information about this variant caller.
get_meta()	Get the metadata about this variant caller.
get_version()	Get the version of the underlying command used.
make_vcf(*args, **kwargs)	Create a VCF from **BAM** file.
validate()	Check if the command can run.

create_aux_files(*ref*)
Create needed (if any) auxiliary files. These files are required for proper functioning of the variant caller.

get_info(*plain=False*)
Get information about this variant caller.

get_meta()
Get the metadata about this variant caller.

get_version()
Get the version of the underlying command used.

make_vcf(**args*, ***kwargs*)
Create a VCF from **BAM** file.

Parameters

ref: str Path to the reference file.

bam: str Path to the indexed **BAM** file for calling _variants.

vcf_file: str path to the VCF file where data will be written to.

make_aux: bool, optional True/False create auxilliary files (default: False).

Returns

bool: True if variant calling was successful, False otherwise.

validate()
Check if the command can run.

class VariantSet(*vcf_in*, *filters=None*, *reference=None*)
Bases: object

A convenient representation of set of _variants. TODO: Implement iterator and generator for the variant set.

Methods

add_metadata(info)	Add metadata to the variant set.
filter_variants([keep_only_snps, only_good])	Filter the VCF records.
variants([only_good])	Generator over the variant set.
write_to_json(vcf_file_name[, verbose])	Write _variants to a json file.
write_variants(vcf_out[, only_snps, only_good])	Write _variants to a VCF file.

add_metadata(*info*)
Add metadata to the variant set.

Parameters

info: dict Dictionary of key value pairs to be inserted into metadata.

filter_variants(*keep_only_snps=False*, *only_good=False*)
Filter the VCF records.

Parameters

keep_only_snps: bool, optional Retain only SNP variants (default: False).

only_good: bool, optional True/False if only SNPs that PASS should output.

Returns

list of records is returned.

variants(*only_good=False*)
Generator over the variant set.

Parameters

only_good: bool, optional Iff True good and bad variants are returned, otherwise only good are returned (default: False).

write_to_json(*vcf_file_name*, *verbose=False*)
Write _variants to a json file.

write_variants(*vcf_out*, *only_snps=False*, *only_good=False*)
Write _variants to a VCF file.

Parameters

vcf_out: str Path to the file where VCF data is written.

only_snps: bool, optional True is *only* SNP are to be written to the output (default: False).

only_good: bool, optional True if only those records that PASS all filters should be written (default: False).

Returns:

int: Number of records written.

phe.variant_filters

Submodules

phe.variant_filters.ADFilter module

Filter VCFs on AD ratio.

Date 24 Sep, 2015

Author Alex Jironkin

class ADFilter(*args*)

Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by AD ratio.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)

hook to extend argparse parser with custom arguments

name = 'ADRatio'

parameter = 'ad_ratio'

short_desc()

Short description of the filter (included in VCF).

phe.variant_filters.DP4Filter module

Filter VCFs on AD ratio.

Date 24 Sep, 2015

Author Alex Jironkin

class DP4Filter(*args*)

Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by DP4 ratio.

Methods

__call__(record)	Filter a vcf.model._Record.
decode(filter_id)	Decode name of filter.
filter_name()	Create filter names by their parameter separated by magic.
get_config()	This is used for reconstructing filter.
is_uncallable(record)	Filter a vcf.model._Record.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)

hook to extend argparse parser with custom arguments

name = 'DP4'

parameter = 'dp4_ratio'

short_desc()

Short description of the filter (included in VCF).

phe.variant_filters.DepthFilter module

Filter VCF on depth of coverage.

Date 24 Sep, 2015

Author Alex Jironkin

class DepthFilter(*args*)

Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by depth.

Methods

__call__(record)	Filter a vcf.model._Record.
decode(filter_id)	Decode name of filter.
filter_name()	Create filter names by their parameter separated by magic.
get_config()	This is used for reconstructing filter.
is_uncallable(record)	Filter a vcf.model._Record.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

name = 'MinDepth'

parameter = 'min_depth'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.GQFilter module

Filter VCF on GQ parameter.

Date 24 Sep, 2015

Author Alex Jironkin

class GQFilter(*args*)
Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by GQ score.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

name = 'MinGQ'

parameter = 'gq_score'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.GTFilter module

Filter VCF on GT filter parameter.

Date 24 Sep, 2015

Author Alex Jironkin

class UncallableGTFilter(*args*)
Bases: *phe.variant_filters.PHEFilterBase*

Filter uncallable genotypes

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

is_gap()

is_n()

name = 'UncallGT'

parameter = 'uncall_gt'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.MQ0FFilter module

Filter VCF on MQ filter.

Date 24 Sep, 2015

Author Alex Jironkin

class MQ0FFilter(*args*)
Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by MQ0F (Total Mapping Quality Zero Reads to DP) ratio.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

name = 'MinMQ0F'

parameter = 'mq0f_ratio'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.MQ0Filter module

Filter VCF on MQ filter.

Date 24 Sep, 2015

Author Alex Jironkin

class MQ0Filter(*args*)
Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by MQ0 (Total Mapping Quality Zero Reads) to DP ratio.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

name = 'MinMQ0'

parameter = 'mq0_ratio'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.MQFilter module

Filter VCF on MQ filter.

Date 24 Sep, 2015

Author Alex Jironkin

class MQFilter(*args*)
Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by Mapping Quality (MQ) score.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)
hook to extend argparse parser with custom arguments

name = 'MinMQ'

parameter = 'mq_score'

short_desc()
Short description of the filter (included in VCF).

phe.variant_filters.QualFilter module

Filter VCF on GQ parameter.

Date 24 Sep, 2015

Author Alex Jironkin

class QualFilter(*args*)

Bases: *phe.variant_filters.PHEFilterBase*

Filter sites by QUAL score.

Methods

`__call__`(record)	Filter a `vcf.model._Record`.
`decode`(filter_id)	Decode name of filter.
`filter_name`()	Create filter names by their parameter separated by magic.
`get_config`()	This is used for reconstructing filter.
`is_uncallable`(record)	Filter a `vcf.model._Record`.

call_concensus	
customize_parser	
is_gap	
is_n	
short_desc	

classmethod customize_parser(*parser*)

hook to extend argparse parser with custom arguments

name = 'LowSNPQual'

parameter = 'qual_score'

short_desc()

Short description of the filter (included in VCF).

Module contents

Classes and functions for working with variant filters.

Implementing Filter

The *PHEFilterBase* class was designed to be implemented in order to provide new filters for processing. As such it should be easy to extend *PHEFilterBase* to achieve this.

If you are not familiar with Python and classes please read this. It is a general introduction to the subject.

Implementation of a filter closely follows the structure for PyVCF

PHEBaseFilter adds additional infomations that is also added to the VCF header:

- *PHEFilterBase.parameter* - Atrribute that allows dynmic loader to pickup the filter.This is the name used in command line arguments and configs.
- PHEFilterBase._default_threshold - Default threshold used for the filter when non specified.
- *PHEFilterBase.short_desc()* - Method for defining what the short description is dynamically.
- PHEFilterBase.__call__() - Function that does the filtering (the same as PyVCF).

The most important of these is the **__call__** function, as this is what does the filtering. It needs to return the following:

- **None** if the record **PASSES** the filter.
- Non **None** expressions are treated as failures.

Note: While it may not be obvious, but Python always returns a value. When no *return* statement is specified, **None** is returned to the calling function.

False is returned when data can not be accessed by the filter.

Once you have implemented the filter and saved it in the *variant_filters* directory, it should be automatically picked up by the system and available to filter on. To verify this run:

```
run_snp_pipeline.py --help
```

You filter should now be visible in the list of available filters.

Example

```
class MyFilter(PHEFilterBase):
    name="CoolQUAL"
    _default_threshold=40
    parameter=cool_qual

    def __init__(self, args):
        # Construct any specific details for the object

    def __call__(self, record):
        if record.QUAL < self.threshold:
            return record.QUAL

    def short_desc(self):
        return "My cool QUAL filter"
```

Date 24 Sep, 2015

Author Alex Jironkin

class PHEFilterBase(*args*)

Bases: vcf.filters.Base

Base class for VCF filters.

Attributes

parameter Short name of parameter being filtered.

Methods

`__call__`()	filter a site, return not None if the site should be filtered
`customize_parser`(parser)	hook to extend argparse parser with custom arguments
decode(filter_id)	Decode name of filter.
filter_name()	Create filter names by their parameter separated by magic.
get_config()	This is used for reconstructing filter.
is_uncallable(record)	Filter a `vcf.model._Record`.
short_desc()	Short description of the filter (included in VCF).

call_concensus	
is_gap	
is_n	

static call_concensus(*record*)

static decode(*filter_id*)
Decode name of filter.

decoder_pattern = <_sre.SRE_Pattern object>

filter_name()
Create filter names by their parameter separated by magic. E.g. if filter parameter is ad_ratio and threshold is 0.9 then ad_ratio:0.9 if the filter name.

get_config()
This is used for reconstructing filter.

is_gap()

is_n()

is_uncallable(*record*)
Filter a `vcf.model._Record`.

magic_sep = ':'

parameter
Short name of parameter being filtered.

short_desc()
Short description of the filter (included in VCF).

available_filters()
Return list of available filters.

dynamic_filter_loader()
Fancy way of dynamically importing existing filters.

Returns

dict: Available filters dictionary. Keys are parameters that can be supplied to the filters.

make_filters(*config*)
Create a list of filters from *config*.

str_to_filters(*filters*)
Convert from filter string to array of filters. E.g. ad_ration:0.9,min_depth:5

Parameters

filters: str String version of filters, separated by comma.

Returns

list: List of *phe.variant_filters.PHEFilterBase* instances.

phe.utils

Submodules

phe.utils.reader

Date 3 June, 2016

Author Public Health England

class ParallelVCFReader(*vcfs*)
Bases: object

Read multiple VCFs in parallel - one position at the time.

Methods

get_records()	Generator of records, one position at the time.
get_samples()	Get the list of sample names from the VCFs.
update([ids])	Update records in the readers.

get_records()
Generator of records, one position at the time. If a position has more than 1 record in any of the readers, a list of all records with that position is returned. The user can deal appropriately with the records list.

Returns

chrom: str Chromosome

pos: int Position

records: dict Records in dictionary of lists for each sample.

```
{
    "sample1": [record1, record2],
    "sample2": [record3]
}
```

get_samples()
Get the list of sample names from the VCFs.

update(*ids=None*)
Update records in the readers. If *ids* is **not** specified, then all readers are updated. Otherwise, only those with the same *id* are updated.

Parameters

ids: list, optional Optional list of IDs to update. If **None**, all will be updated.

Module contents

class BaseStats
Bases: object

Methods

update	

update(*position_data*, *sample*, *reference*)

calculate_memory_for_sort()
Calculate available memory for samtools sort function. If there is enough memory, no temp files are created. **Enough** is defined as at least 1G per CPU.

Returns

sort_memory: str or None String to use directly with *-m* option in sort, or None.

getTotalNofDiff_tn84(*d*)
Sum up total number of differences for a dict like the one in the input

Parameters

d: dict

{'A': {'A': 1152.0, 'C': 114.0, 'G': 545.0, 'T': 35.0}, 'C': {'A': 0.0, 'C': 1233.0, 'G': 108.0, 'T': 467.0}, 'G': {'A': 0.0, 'C': 0.0, 'G': 1283.0, 'T': 100.0}, 'T': {'A': 0.0, 'C': 0.0, 'G': 0.0, 'T': 1177.0}}

Returns

t: float the sum of all differences(1369.0 in above example case)

get_difference_value(*s1_base*, *s2_base*, *sSubs*)
Get difference value for a given set of bases.

Parameters

s1_base: str a charcater

s2_base: str a charcater

sSubs: str distance model

Returns

difference: float or list depending on the distance model either a float 1.0 or 0.0 of a list of two floats

get_dist_mat(*aSampleNames*, *avail_pos*, *dArgs*)
Calculates the distance matrix, optionally removes recombination from it and optionally normalises it

Parameters

aSampleNames: list list of sample names

avail_pos: dict infomatin on all available positions {'gi|194097589|ref|NC_011035.1|':

FastRBTree({2329: {'stats': <vcf2distancematrix.BaseStats object at 0x40fb590>,

'reference': 'A', '211700_H15498026501': 'C', '211701_H15510030401': 'C', '211702_H15522021601': 'C'},

3837: {'211700_H15498026501': 'G', 'stats': <vcf2distancematrix.BaseStats object at 0x40fbf90>, '211701_H15510030401': 'G', 'reference': 'T', '211702_H15522021601': 'G'},

4140: {'211700_H15498026501': 'A', 'stats': <vcf2distancematrix.BaseStats object at 0x40fb790>, '211701_H15510030401': 'A', 'reference': 'G', '211702_H15522021601': 'A'}})}

dArgs: dict input parameter dictionary as created by get_args()

Returns

call to get_sample_pair_densities with parameters unpacked

get_ref_freqs(*ref*, *len_only=False*)

Get the length of the reference genome and optionally the nucleotide frequencies in it

Parameters

ref: str reference genome filename

len_only: boolean get genome lengths only [default FALSE, also get nucleotide frequencies]

Returns

(dRefFreq, flGenLen): tuple

dRefFreq: dict dRefFreq = {'A': 0.25, 'C': 0.24, 'G': 0.26, 'T': 0.25}

flGenLen: float genome length

get_sample_pair_densities(*sample_1*, *sample_2*, *oBT*, *flGenLen*)

Function to calculate the differecnes in a window of a given size around is difference for a given pair

Parameters

sample_1: str name of sample 1

sample_2: str name of sample 2

oBT: obj bintree object that contains all information for all available positions for a given contig

flGenLen: float reference genome length

Returns

——-

(diffs, d): tuple

diffs: int total number of differences between the pair

d: dict dict with position of difference as key and number of differences in window around it as value

is_uncallable(*record*)

Is the Record uncallable? Currently the record is **uncallable** iff:

- GT field is **./.**

- **LowQual** is in the filter.

Returns

uncall: bool True if any of the above items are true, False otherwise.

normalise_jc69(*d*, *ref*, *names*)

Normalise distance matrix according to the Jukes-Cantor distance model see: Nei and Zhang: Evolutionary Distance: Estimation,

ENCYCLOPEDIA OF LIFE SCIENCES 2005, doi: 10.1038/npg.els.0005108 http://www.umich.edu/~zhanglab/publications/2003/a0005108.pdf, equation 7

Parameters

d: dict distance matrix

ref: str reference genome file name

names: list list of sample names

Returns

——-

d: dict normalised matrix

normalise_k80(*d*, *ref*, *names*)

Normalise distance matrix according to the Tajima-Nei distance model see: Nei and Zhang: Evolutionary Distance: Estimation,

ENCYCLOPEDIA OF LIFE SCIENCES 2005, doi: 10.1038/npg.els.0005108 http://www.umich.edu/~zhanglab/publications/2003/a0005108.pdf, equation 9

Parameters

d: dict distance matrix

ref: str reference genome file name

names: list list of sample names

Returns

——-

d: dict normalised matrix

normalise_t93(*d*, *ref*, *names*)

Normalise distance matrix according to the Tamura 3-parameter distance model see: Nei and Zhang: Evolutionary Distance: Estimation,

ENCYCLOPEDIA OF LIFE SCIENCES 2005, doi: 10.1038/npg.els.0005108 http://www.umich.edu/~zhanglab/publications/2003/a0005108.pdf, equation 16

Parameters

d: dict distance matrix

ref: str reference genome file name

names: list list of sample names

Returns

———-

d: dict normalised matrix

normalise_tn84 (*d*, *ref*, *names*)

Normalise distance matrix according to the Kimura 2-parameter distance model

see: Nei and Zhang: Evolutionary Distance: Estimation, ENCYCLOPEDIA OF LIFE SCIENCES 2005, doi: 10.1038/npg.els.0005108 http://www.umich.edu/~zhanglab/publications/2003/a0005108.pdf, equation 13 and 14

Parameters

d: dict distance matrix d = {'211701_H15510030401': {'211700_H15498026501': {'A': {'A': 1152.0,

'C': 114.0, 'G': 545.0, 'T': 35.0},

'C': {'A': 0.0, 'C': 1233.0, 'G': 108.0, 'T': 467.0},

'G': {'A': 0.0, 'C': 0.0, 'G': 1283.0, 'T': 100.0},

'T': {'A': 0.0, 'C': 0.0, 'G': 0.0, 'T': 1177.0}}, . . . }, . . . }

ref: str reference genome file name

names: list list of sample names

Returns

———-

d: dict normalised matrix

parse_vcf_files (*dArgs*, *avail_pos*, *aSampleNames*)

Parse vcf files to data structure Parameters ————- dArgs: dict

input parameter dictionary as created by get_args()

avail_pos: dict dict of bintrees for each contig

aSampleNames: list list of sample names

0 also writes all data to avail_pos

parse_wg_alignment (*dArgs*, *avail_pos*, *aSampleNames*)

Parse alignment to data structure Parameters ————- dArgs: dict

input parameter dictionary as created by get_args()

avail_pos: dict dict of bintrees for each contig

aSampleNames: list list of sample names

0 also writes all data to avail_pos

precompute_snp_densities (*avail_pos*, *sample_names*, *args*)

Precompute the number of differences around each difference between each pair of samples

Parameters

avail_pos: dict data structure that contains the information on all available positions, like this: {'gi|194097589|ref|NC_011035.1|': FastRBTree({2329: {'stats': <vcf2distancematrix.BaseStats object at 0x40fb590>,

'reference': 'A', '211700_H15498026501': 'C', '211701_H15510030401': 'C', '211702_H15522021601': 'C'},

3837: {'211700_H15498026501': 'G', 'stats': <vcf2distancematrix.BaseStats object at 0x40fbf90>, '211701_H15510030401': 'G', 'reference': 'T', '211702_H15522021601': 'G'},

4140: {'211700_H15498026501': 'A', 'stats': <vcf2distancematrix.BaseStats object at 0x40fb790>, '211701_H15510030401': 'A', 'reference': 'G', '211702_H15522021601': 'A'}})}

sample_names: list list of sample names

args: dict input parameter dictionary as created by get_args()

Returns

dDen: dict contains the differences between a pair in a window of given size around each difference of the pair {'diffs': {'187534_H153520399-1': {'187534_H153520399-1': 0,

'187536_H154060132-1': 1609, '189918_H154320283-2': 295, '205683_H15352039901': 0, '211698_H15464036401': 298, '211700_H15498026501': 298, '211701_H15510030401': 1621, '211702_H15522021601': 297, '211703_H15534021301': 1632, 'reference': 4045},

'187536_H154060132-1': {'187536_H154060132-1': 0,
'205683_H15352039901': 1605, '211698_H15464036401': 1353, '211701_H15510030401': 1, '211702_H15522021601': 1351, '211703_H15534021301': 7, 'reference': 5041} . . . , },

'gi|194097589|ref|NC_011035.1|': {'187534_H153520399-1': {'187536_H154060132-1': {55959: 1,

56617: 1, 157165: 1, 279950: 3, 279957: 3, 279959: 3, 608494: 22, 608537: 23, 608551: 23, 608604: 23, 608617: 24, . . . ,}

'189918_H154320283-2': {27696: 1,

55959: 1, 56617: 2, 56695: 2, 279950: 3, 279957: 3, 279959: 3, 520610: 1, 608494: 22, . . . ,

Module contents

CHAPTER 8

Scripts

Useful scripts for data analysis.

8.1 phenix

There is a single entry point to access all commands we have produced so far. Please refer to this documentation or *–help* on the cammand line.

```
usage: phenix [-h] [--debug] [--version]
              {run_snp_pipeline,filter_vcf,prepare_reference,vcf2fasta,
↪vcf2distancematrix,vcf2json}
              ...
```

8.1.1 Positional Arguments

cmd Possible choices: run_snp_pipeline, filter_vcf, prepare_reference, vcf2fasta, vcf2distancematrix, vcf2json

8.1.2 Named Arguments

--debug More verbose logging (default: turned off).

Default: False

--version show program's version number and exit

8.1.3 Sub-commands:

run_snp_pipeline

Run the snp pipeline with specified mapper, variant caller and some filters.

Available mappers: ['bwa', 'bowtie2']

Available variant callers: ['mpileup', 'gatk']

Available filters: ['gq_score', 'dp4_ratio', 'ad_ratio', 'min_depth', 'mq_score', 'mq0_ratio', 'uncall_gt', 'qual_score', 'mq0f_ratio']

Available annotators: ['coverage']

```
phenix run_snp_pipeline [-h] [--workflow WORKFLOW] [--input INPUT] [-r1 R1]
                        [-r2 R2] [--reference REFERENCE]
                        [--sample-name SAMPLE_NAME] [--outdir OUTDIR]
                        [--config CONFIG] [--mapper MAPPER]
                        [--mapper-options MAPPER_OPTIONS] [--bam BAM]
                        [--variant VARIANT]
                        [--variant-options VARIANT_OPTIONS] [--vcf VCF]
                        [--filters FILTERS]
                        [--annotators ANNOTATORS [ANNOTATORS ...]]
                        [--keep-temp] [--json] [--json-info]
```

Named Arguments

--workflow, -w	
--input, -i	
-r1	R1/Forward read in Fastq format.
-r2	R2/Reverse read in Fastq format.
--reference, -r	Rerefence to use for mapping.
--sample-name	Name of the sample for mapper to include as read groups.
	Default: "test_sample"
--outdir, -o	
--config, -c	
--mapper, -m	Available mappers: ['bwa', 'bowtie2']
	Default: "bwa"
--mapper-options	Custom maper options (advanced)
--bam	
--variant, -v	Available variant callers: ['mpileup', 'gatk']
	Default: "gatk"
--variant-options	Custom variant options (advanced)
--vcf	

--filters	Filters to be applied to the VCF in key:value pairs, separated by comma (,). Available_filters: ['gq_score', 'dp4_ratio', 'ad_ratio', 'min_depth', 'mq_score', 'mq0_ratio', 'uncall_gt', 'qual_score', 'mq0f_ratio']. Recommendations: GATK: mq_score:30,min_depth:10,ad_ratio:0.9 Mpileup: mq_score:30,min_depth:10,dp4_ratio:0.9
--annotators	List of annotators to run before filters. Available: ['coverage']
--keep-temp	Keep intermediate files like BAMs and VCFs (default: False). Default: False
--json	Also write variant positions in filtered vcf as json file (default: False). Default: False
--json-info	When writing a json file, log some stats to stdout. (default: False). Default: False

filter_vcf

Filter the VCF using provided filters.

```
phenix filter_vcf [-h] --vcf VCF [--filters FILTERS | --config CONFIG]
                  --output OUTPUT [--reference REFERENCE] [--only-good]
```

Named Arguments

--vcf, -v	VCF file to (re)filter.
--filters, -f	Filter(s) to apply as key:threshold pairs, separated by comma. Recommendations: GATK: mq_score:30,min_depth:10,ad_ratio:0.9 Mpileup: mq_score:30,min_depth:10,dp4_ratio:0.9
--config, -c	Config with filters in YAML format. E.g.filters:-key:value
--output, -o	Location for filtered VCF to be written.
--reference, -r	mpileup version <= 1.3 do not output all positions. This is required to fix rfrence base in VCF.
--only-good	Write only variants that PASS all filters (default all variants are written). Default: False

prepare_reference

Prepare reference for SNP pipeline by generating required aux files.

```
phenix prepare_reference [-h] --reference REFERENCE [--mapper MAPPER]
                         [--variant VARIANT]
```

Named Arguments

--reference, -r Path to reference file to prepare.

--mapper Available mappers: [‘bwa’, ‘bowtie2’]

--variant Available variants: [‘mpileup’, ‘gatk’]

vcf2fasta

Combine multiple VCFs into a single FASTA file.

```
phenix vcf2fasta [-h] (--directory DIRECTORY | --input INPUT [INPUT ...])
                 [--regexp REGEXP] --out OUT [--with-mixtures WITH_MIXTURES]
                 [--column-Ns COLUMN_NS] [--column-gaps COLUMN_GAPS]
                 [--sample-Ns SAMPLE_NS] [--sample-gaps SAMPLE_GAPS]
                 [--sample-Ns-gaps-auto-factor SAMPLE_NS_GAPS_AUTO_FACTOR]
                 [--reference REFERENCE | --remove-invariant-npos]
                 [--reflength REFLENGTH]
                 [--include INCLUDE | --exclude EXCLUDE]
                 [--with-stats WITH_STATS]
```

Named Arguments

--directory, -d Path to the directory with .vcf files.

--input, -i List of VCF files to process.

--regexp Regular expression for finding VCFs in a directory.

--out, -o Path to the output FASTA file.

--with-mixtures Specify this option with a threshold to output mixtures above this threshold.

--column-Ns Keeps columns with fraction of Ns below specified threshold.

--column-gaps Keeps columns with fraction of Ns below specified threshold.

--sample-Ns Keeps samples with fraction of Ns below specified threshold or put ‘auto’.Fraction expressed as fraction of genome. Requires –reflength or –reference.

--sample-gaps Keeps samples with fraction of gaps below specified threshold or put ‘auto’.Fraction expressed as fraction of genome. Requires –reflength or –reference.

--sample-Ns-gaps-auto-factor When using ‘auto’ option for –sample-gaps or –sample-Ns, remove sample that havegaps or Ns this many times above the stddev of all samples. [Default: 2.0]

Default: 2.0

--reference If path to reference specified (FASTA), then whole genome will be written to alignment.

--remove-invariant-npos Remove all positions that invariant apart from N positions.

Default: False

--reflength	Length of reference. Either as int or can be worked out from fasta file. Ignored if –reference is used.
--include	Only include positions in BED file in the FASTA
--exclude	Exclude any positions specified in the BED file.
--with-stats	If a path is specified, then position of the outputed SNPs is stored in this file.

vcf2distancematrix

Combine multiple VCFs into a distance matrix. Distance measures according to five different models are available: * Number of differences

- Jukes-Cantor distance (jc69)
- Tajima-Nei distance (k80)
- Kimura 2-parameter distance (tn84)
- Tamura 3-parameter distance (t93)

```
phenix vcf2distancematrix [-h]
                          (--directory DIRECTORY | --input INPUT [INPUT ...] | --
↪alignment-input MULTI FASTA FILE)
                         --out OUT [--deletion STRING]
                         [--substitution STRING]
                         [--include BED FILE | --exclude BED FILE]
                         [--remove-recombination] [--refgenome FASTA FILE]
                         [--refgenomename STRING] [--threshold FLOAT]
                         [--threads INT] [--format STRING] [--tree FILE]
                         [--with-stats]
```

Named Arguments

--directory, -d	Path to the directory with .vcf files. Input option 1.
--input, -i	List of VCF files to process. Input option 2.
--alignment-input, -a	Multi fasta file with whole genome input alignment. Input option 3.
--out, -o	Path to the maxtrix output file in given format. [REQUIRED. default format is tab separated. use –format to change format]
--deletion	Possible choices: pairwise, complete Method of recombination filtering. Either ‘pairwise’ or ‘complete’ [‘pairwise’] Default: “pairwise”
--substitution	Possible choices: number_of_differences, jc69, k80, tn84, t93 Substituition model. Either ‘number_of_differences’, ‘jc69’, ‘k80’, ‘tn84’ or ‘t93’ [‘number_of_differences’] Default: “number_of_differences”
--include	Only include positions in BED file in the FASTA
--exclude	Exclude any positions specified in the BED file.

--remove-recombination Attempt to remove recombination from distance matrix. [don't]

Default: False

--refgenome, -g Reference genome used for SNP calling [Required for recombination removal].

--refgenomename, -n Name of reference genome in input alignment [Required if input option 3 is used and reference is not named 'reference'].

--threshold, -k Density tyhreshold above mean density for relevant pair. [1.0].

Default: 1.0

--threads Number of threads to use. [1].

Default: 1

--format Possible choices: tsv, csv, mega

Change format for output file. Available options csv, tsv and mega. [tsv]

Default: "tsv"

--tree, -t Make an NJ tree and write it to the given file in newick format. [Default: Don't make tree, only matrix]

--with-stats Write additional files with information on removed recombinant SNPs. [don't]

Default: False

vcf2json

Converts the postions of variants and ignored/missing positions in either a 'raw' or filtered VCF file to a json string and writes it to a file. The json contains 6 arrays for each chromosome in the VCF file: g_positions, a_positions, t_positions, c_positions, gap_positions, n_positions

```
phenix vcf2json [-h] --input INPUT [--output_file_prefix OUTPUT_FILE_PREFIX]
                [--nozip] [--vcf_is_filtered] [--summary_info]
```

Named Arguments

--input, -i path to a VCF file

--output_file_prefix, -o Path to the json output file (without file extension). Default: sample_name

Default: "sample_name"

--nozip, -z Do not gzip json when writing file. (default: Yes, gzip it.)

Default: False

--vcf_is_filtered, -f Required: Confirm that the input vcf is filtered. It is stronglyrecommended to filter the file with Phenix using the sameparameters that are used throughout the database this jason file is meant for.

Default: False

--summary_info, -s Print summary of the json string

Default: False

CHAPTER 9

Galaxy

This tools is available for installation to your local Galaxy [http://galaxyproject.org] from the Toolshed [https://toolshed.g2.bx.psu.edu/]. This section describes in some detail how this is done and how the Galaxy tool can be used to process data.

9.1 Primer

The Phenix Galaxy tool only enables you to filter VCF files and to convert VCF files into FASTA files. The mapping and the SNP calling component of Phenix need to be provided by other Galaxy tools (also available from the Toolshed.)

9.2 How to get your own Galaxy server:

If you already have a local Galaxy server on which you are an Administrator and on which you can install tools from the Toolshed you can skip this section and continue reading at "How to install Phenix to your Galaxy server". If not you might want to get one, which is easy.

Prerequisites:

- A workstation running a Linux operating system
- Min. 4GB of RAM (better 8GB or more)
- Root access to that machine.
- git (optional)
- Python 2.6 or 2.7
- GNU Make, gcc to compile and install tool dependencies

The last three are standard on most contemporary Linux installations.

Get your own Galaxy:

Please go to [https://wiki.galaxyproject.org/Admin/GetGalaxy] and follow the instructions in the sections from "Get the Code" to "Become an Admin" (including).

9.3 How to install Phenix to your Galaxy server:

This page (https://wiki.galaxyproject.org/Admin/Tools/AddToolFromToolShedTutorial) describes the general procedure of installing a tool from the toolshed. Here is a click-by-click guide to what you need to do to install Phenix.

Prerequisites:

There is a small number of Linux packages that need to be installed on your machine, so that Phenix can be installed and run propery. These are:

- a C++ compiler
- a Fortran compiler
- the zlib development libraries
- the samtools package

The exact name of these packages and the commands to install them depend on yout Linux distribution. For Ubuntu, Mint and Debian Linux the required commands should be:

```
sudo apt-get install build-essential
sudo apt-get install gfortran
sudo apt-get install zlib-devel
sudo apt-get install samtools
```

On Fedora the commands should be:

```
sudo dnf install gcc-c++
sudo dnf install gcc-gfortran
sudo dnf install zlib-devel
sudo dnf install samtools
```

And on OpenSUSE the commands should be:

```
sudo zypper install gcc-c++
sudo zypper install gcc-fortran
sudo zypper install zlib-devel
sudo zypper install samtools
```

For more esoteric Linux distributions please refer to your local IT expert and/or Google. After you have successfully installed these packages, follow the instructions below.

- Make sure that you can access ftp sites from the command line running your Galaxy server. This is normally enabled by default, but sometimes requires an additional proxy setting. Try this command

```
wget ftp://ftp.gnu.org/gnu/libtool/libtool-2.4.tar.gz
```

If that does not download a file named 'libtool-2.4.tar.gz' to your current folder, speak to your local systems' administrator.

- In your config/galaxy.ini files, set

```
tool_dependency_dir = tool_dependencies
```

- Restart your Galaxy

- In your Galaxy, click on 'Admin' in the main menu at the top.
- Select 'Search Tool Shed' from the menu on the left hand side.
- Click on the little black triangle (the context menu) next to 'Galaxy Main Tool Shed' and select 'Browse valid repositories'.
- Type 'phephenix' [sic] into the search box and press Enter.
- You should see the "package_phephenix_1_0" and the "phephenix" repositories. Select 'Preview and install' from the context menu of the latter.
- Click 'Install to Galaxy'.
- Type 'PHE TOOLS' into the 'Add new tool panel section:' textbox.
- Click 'Install'.
- You will be presented with a long list of packages that need to be installed. This will take a while. Wait until everything is green. If nothing happens for a little while, try reloading the page.

In your admin tool panel the menu item "Manage installed tools" was added. You can check the status of your installed packages there.

You need to install two more tools that are part of the Phenix workflow:

- Browse the toolshed as before and look for 'bwa'. Install the tool 'bwa' with synopsis "Wrapper for bwa mem, aln, sampe, and samse" owned by 'devteam'. Put it into the 'PHE TOOLS' tool panel section.
- Browse the toolshed as before and install the tool 'phe_samtools_mpileup'. Put it into the 'PHE TOOLS' tool panel section.

9.4 How to use Phenix on Galaxy:

- In the admin menu, go the 'Manage installed tools'
- Find the 'phephenix' tool and click on it, not on it's context menu.
- In the 'Contents of this repository' box at the bottom of the page, expand the workflows section using the little blue triange if necessary.
- Click on 'Phenix workflow' and behold an image representation of the workflow.
- Click on 'Repository actions' -> 'Import workflow to Galaxy'

The Phenix workflow is now ready to use. You need to upload your data to a Galaxy history to use it. There are multiple options depending on your local Galaxy configuration. If you have followed the instructions above under 'Get your own Galaxy' the only available option is uploading from your local harddrive. When doing this, please make sure your fastq files are in the 'fastqsanger' format and your reference genome is in 'fasta' format. The basename of your R1 fastq files will be used as the name for the resulting bam and vcf files with appropriate file endings and will also appear as the sequence header in your resulting SNP alignment. Once your data is ready to use follow theses instructions to run the workflow.

Note: We have chosen reasonable defaults for filtering your VCF files for high-confidence SNPs (min_depth:5,mq_score:30,qual_score:30,dp4_ratio:0.9,mq0f_ratio:0.1). If you would like to change these settings please see the instructions below.

- Click on workflow in the top main menu and select 'run' from the 'Phenix workflow' context menu.
- Select your reference genome in the 'Reference fasta' selection box.

- Click both "paper stack" icons next to 'R1 fastqs' and next to 'R2 fastqs'.
- Select all R1.fastq files in the "R1 fastqs" selection box that appeared. An easy way to do this is to type 'R1' into the text box immediately under the selection box. The selection box will automatically be filtered for fastq file names that match what has been typed into the text box.
- Do as above for the "R2 fastq" box.

Note: Make sure that the order of highlighted files in the R1 and R2 fastq selection boxes are corresponding, i.e. the first highlighted files in the top box will be run together with the first highlighted file in the bottom box, the second with the second and so on. Are the highlighted files in the same rank in both boxes always matching pairs from the same sample?

- If you are happy with the above, click 'Run workflow'.
- Wait until three new history items per sample processed appear. These will be one bam file, one raw vcf, and one filtered vcf per sample.
- Click on PHE TOOLS in the tool panel on the left hand side and select the "VCFs to fasta" tool. This tool will create a multi-fasta file from your filtered vcfs.
- Select all files ending with filtered.vcf in the top file selection box under 'Input VCF files(s)', by holding down the Ctrl key.
- Click 'Execute'. Once everything is completed the "VCFs to fasta" tool with have produced your SNP alignment that can now be used for further processing.

How to change the number of jobs running simultaneously?

Galaxy runs 5 jobs at the same time by default. This is appropriate for machines that have at least 5 GB of RAM and at least 5 processor cores. If you have more or less compute resources at your disposal you can change the number of concurrent jobs, if you rename the file *config/job_conf.xml.sample_basic* to *config/job_conf.xml* and change the "workers" setting in the file to the desired number. This requires a restart of Galaxy.

9.5 Advanced - Changing the Phenix vcf filtering settings:

- Click on workflow in the top main menu and select 'edit' from the 'Phenix workflow' context menu.
- Click on the 'Filter VCF' tool box to highlight it.
- The parameters for this tool can now be edited in the panel on the right hand side of the browser window.
- Use the little trash can in the top right corner of each individual filter to remove it.
- Use the button labelled '+ Insert SNP filter' to add a new one.
- Use the drop-down menu to select a new filter type and the corresponding text box to set the threshold value.

Warning: Not all filters are suitable for all variant callers. Please refer to the table under *Filters*.

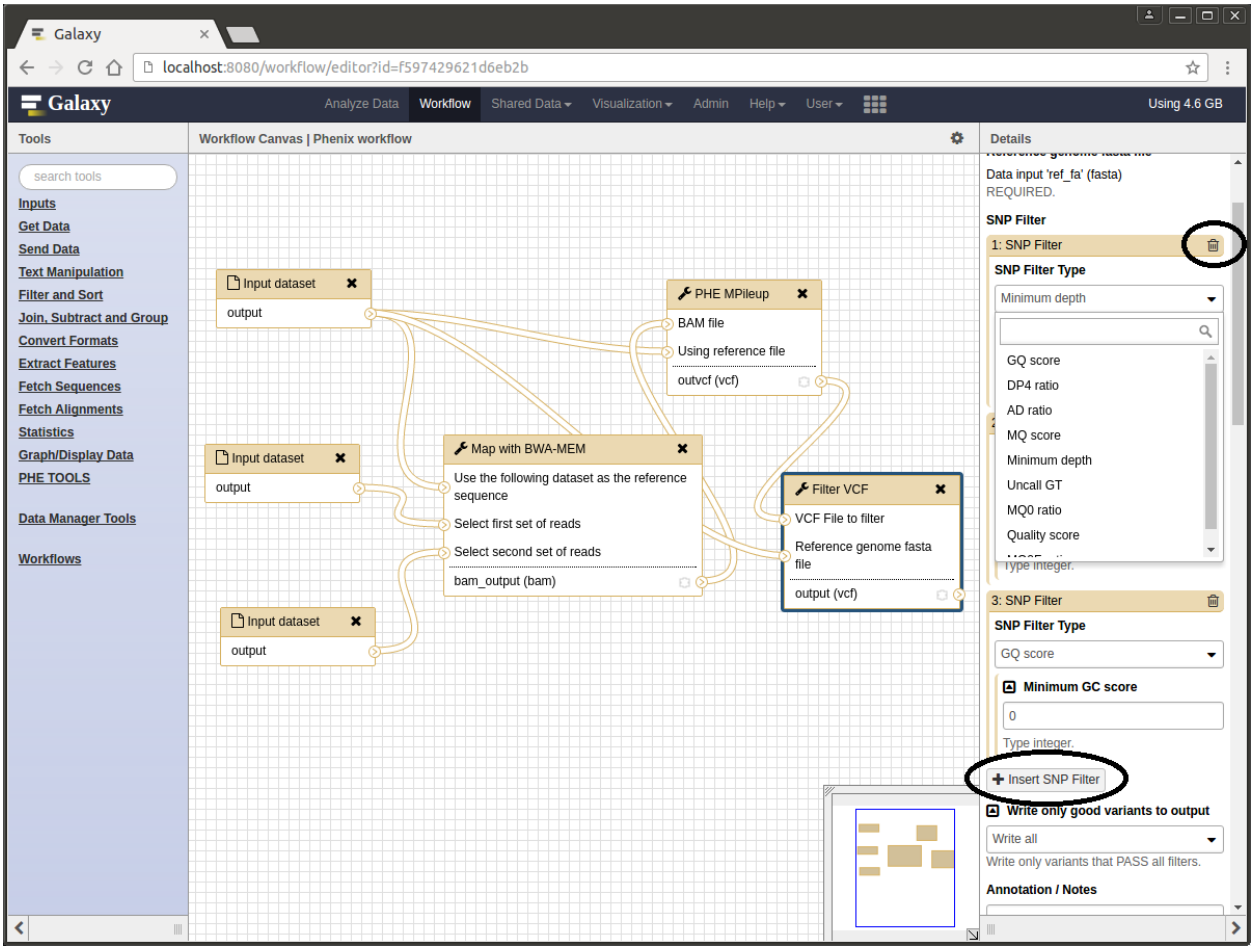

Fig. 1: The Galaxy workflow editor.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

I

M

N

P

Q

S

U

V

W