igsr_analysis Documentation

Release 0.1.1

ernesto lowy

Sep 17, 2020

Contents

1 Installation **3**

2 Workflows **5**
- 2.1 WGS BAM qc pipeline . . . 5
- 2.2 WES BAM qc pipeline . . . 8
- 2.3 BCFtools WGS variant filtering pipeline . . . 10
- 2.4 BCFtools WES variant filtering pipeline . . . 13
- 2.5 Freebayes variant filtering pipeline . . . 15
- 2.6 GATK variant filtering pipeline . . . 17
- 2.7 VCF decoration . . . 20
- 2.8 ML-based workflow to filter a VCF . . . 22
- 2.9 Benchmarking with a true set . . . 23
- 2.10 Consensus call set generation . . . 25
- 2.11 Phasing pipeline . . . 28
- 2.12 CRAM to BAM conversion . . . 30

3 Auto-documentation **33**
- 3.1 VCF . . . 33
- 3.2 BEDTools . . . 34
- 3.3 BamQC . . . 35
- 3.4 VariantCalling . . . 39
- 3.5 Utils . . . 39

Python Module Index **41**

Index **43**

This is the documentation showing how to use the code and how to run the different pipelines we use for the analysis of IGSR data.

The International Genome Sample Resource (IGSR) is a project funded by the Wellcome Trust created after the finalization of the 1000 Genomes Project in order to maintain and expand the resource. It has the following aims:

- Ensure the future access to and usability of the 1000 Genomes reference data
- Incorporate additional published genomic data on the 1000 Genomes samples
- Expand the data collection to include new populations not represented in the 1000 Genomes Project

Contents:

CHAPTER 1

Installation

Preparing environment

Modify your $PYTHONPATH to include the required libraries:

```
export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
```

Modify your $PERL5LIB to include the required libraries:

```
export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
```

Modify your $PATH to include the location of the eHive scripts:

```
export PATH=${ehive_dir}/scripts/:${PATH}
```

Install dependency

1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder
2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.90.tar.gz`

And you are ready to go!

Conventions used in this README file:

- `${igsr_analysis_dir}` is the folder where you have cloned https://github.com/igsr/igsr_analysis.git
- `${ehive_dir}` is the folder where you have cloned https://github.com/Ensembl/ensembl-hive.git

CHAPTER 2

Workflows

This document explains how to run the different workflows used in our project

Contents:

Contents:

2.1 WGS BAM qc pipeline

Workflow used in IGSR to assess the quality of a certain file in the BAM format produced in a Whole Genome Sequencing experiment. This workflow consists on running 3 different types of tests:

- Chkindel_rg

This test consists on using a simple algorithm to identify runs with unbalanced ratio of short insertion and deletion (greater than 5), which is indicative of low quality data. The code to run this test can be found at:

https://github.com/lh3/samtools-legacy/blob/master/examples/chk_indel.c

- VerifyBAMID

This test is used to assess sample contamination and sample mix-ups, it uses the VerifyBAMID software. More info on this useful piece of software can be found at:

https://genome.sph.umich.edu/wiki/VerifyBamID

- Coverage

For assessing the coverage we use Picard CollectWgsMetrics. This software generates a complete report on the depth of coverage of the sequencing experiment and the calculated mean coverage value can be used to decide which files to discard. More information on this Picard tool can be found at:

https://broadinstitute.github.io/picard/command-line-overview.html#CollectWgsMetrics

In order to run this workflow we need to do the following:

1. Preparing the environment

Modify your $PYTHONPATH to include the required libraries:

```
export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
```

Modify your $PERL5LIB to include the required libraries:

```
export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
```

Modify your $PATH to include the location of the eHive scripts:

```
export PATH=${ehive_dir}/scripts/:${PATH}
```

- Install dependency
  1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder
  2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`
  3) Modify $PYTHONPATH to add the folder where your pip installs the Python packages

  And you are ready to go!
- Conventions used in this section
  - ${igsr_analysis_dir} is the folder where you have cloned https://github.com/igsr/igsr_analysis.git
  - ${ehive_dir} is the folder where you have cloned https://github.com/Ensembl/ensembl-hive.git

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a ReseqTrack database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a ReseqTrack database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                         # is the name you want                   ␣
                                                                    ␣
↪                         # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  $RESEQTRACK is the folder where you have cloned https://github.com/EMBL-EBI-GCA/reseqtrack.git

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

   The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::QC::RunBamQCsonWGS \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

   The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

   Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

   The rest of the options are documented in the PyHive::PipeConfig::QC::RunBamQCsonWGS module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                 -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                 -logic_name find_files \
                 -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

Note the '&' makes it run in the background.

Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.2 WES BAM qc pipeline

Workflow used in IGSR to assess the quality of a certain file in the BAM format produced in a Whole Exome Sequencing (WES) experiment. This workflow consists on running 3 different types of tests:

- Chkindel_rg

This test consists on using a simple algorithm to identify runs with unbalanced ratio of short insertion and deletion (greater than 5), which is indicative of low quality data. The code to run this test can be found at:

https://github.com/lh3/samtools-legacy/blob/master/examples/chk_indel.c

- VerifyBAMID

This test is used to assess sample contamination and sample mix-ups, it uses the VerifyBAMID software. More info on this useful piece of software can be found at:

https://genome.sph.umich.edu/wiki/VerifyBamID

- Coverage

For assessing the coverage we use Picard CollectHsMetrics. This software generates a complete report on the depth of coverage for the targeted regions from the WES. More information on this Picard tool can be found at:

https://broadinstitute.github.io/picard/command-line-overview.html#CollectHsMetrics

In order to run this workflow we need to do the following:

1. Preparing the environment

   Modify your `$PYTHONPATH` to include the required libraries:

   ```
   export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
   ```

   Modify your `$PERL5LIB` to include the required libraries:

   ```
   export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
   ```

   Modify your `$PATH` to include the location of the eHive scripts:

   ```
   export PATH=${ehive_dir}/scripts/:${PATH}
   ```

   - Install dependency

     1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder

     2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`

3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

And you are ready to go!

- Conventions used in this section
  - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
  - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a `ReseqTrack` database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                  ␣
↪                         # is the name you want
                                                                  ␣
↪                         # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::QC::RunBamQCsonWES \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::QC::RunBamQCsonWES module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                -logic_name find_files \
                -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

   Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

   While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.3 BCFtools WGS variant filtering pipeline

In order to run this workflow we need to do the following:

1. Preparing the environment

Modify your `$PYTHONPATH` to include the required libraries:

```
export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
```

Modify your `$PERL5LIB` to include the required libraries:

```
export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
```

Modify your `$PATH` to include the location of the eHive scripts:

```
export PATH=${ehive_dir}/scripts/:${PATH}
```

- Install dependency
  1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder
  2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`
  3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

  And you are ready to go!
- Conventions used in this section
  - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
  - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a `ReseqTrack` database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                          # is the name you want                   ␣
                                                                    ␣
↪                          # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

   The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::FILTER::VCFilterSamtoolsWGS \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

   The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

   Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

   The rest of the options are documented in the PyHive::PipeConfig::FILTER::VCFilterSamtoolsWGS module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                 -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                 -logic_name find_files \
                 -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

Note the '&' makes it run in the background.

Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.4 BCFtools WES variant filtering pipeline

In order to run this workflow we need to do the following:

1. Preparing the environment

   Modify your `$PYTHONPATH` to include the required libraries:

   ```
   export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
   ```

   Modify your `$PERL5LIB` to include the required libraries:

   ```
   export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
   ```

   Modify your `$PATH` to include the location of the eHive scripts:

   ```
   export PATH=${ehive_dir}/scripts/:${PATH}
   ```

   - Install dependency

     1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder

     2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`

     3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

     And you are ready to go!

   - Conventions used in this section
     - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
     - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

2. Databases

   The pipeline uses two databases. They may be on different servers or the same server.

   2.1 The ReseqTrack database

   The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

   In order to create a `ReseqTrack` database use the following

   commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                          # is the name you want
                                                                    ␣
↪                          # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::FILTER::VCFilterSamtoolsWES \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::FILTER::VCFilterSamtoolsWES module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                 -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                 -logic_name find_files \
                 -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

   Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

   While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.5 Freebayes variant filtering pipeline

In order to run this workflow we need to do the following:

1. Preparing the environment

   Modify your `$PYTHONPATH` to include the required libraries:

```
export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
```

   Modify your `$PERL5LIB` to include the required libraries:

```
export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
```

   Modify your `$PATH` to include the location of the eHive scripts:

```
export PATH=${ehive_dir}/scripts/:${PATH}
```

   - Install dependency

     1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder

     2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`

     3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

     And you are ready to go!

   - Conventions used in this section

- ${igsr_analysis_dir} is the folder where you have cloned https://github.com/igsr/igsr_analysis.git
- ${ehive_dir} is the folder where you have cloned https://github.com/Ensembl/ensembl-hive.git

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a ReseqTrack database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a ReseqTrack database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                          # is the name you want
                                                                    ␣
↪                          # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  $RESEQTRACK is the folder where you have cloned https://github.com/EMBL-EBI-GCA/reseqtrack.git

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the init_pipeline.pl script. Write access is needed to this database.

3. Initialise the pipeline

The pipeline is initialised with the hive script init_pipeline.pl. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::FILTER::VCFilterFreeBayes \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then -pipeline_url controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then -db is the name of the Reseqtrack database name used in the section 2.1 -pwd is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::FILTER::VCFilterFreebayes.pm module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script seed_pipeline.pl:

```
seed_pipeline.pl \
                -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                -logic_name find_files \
                -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where -url controls the Hive database connection details and /path/to/file/input_file.txt contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where -url are the details of your hive database. Look at the output from init_pipeline.pl to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

   Look at the pod for beekeeper.pl to see the various options. E.g. you might want to use the -hive_log_dir flag so that all output/error gets recorded in files.

   While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.6 GATK variant filtering pipeline

In order to run this workflow we need to do the following:

1. Preparing the environment

   Modify your $PYTHONPATH to include the required libraries:

```
export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
```

   Modify your $PERL5LIB to include the required libraries:

```
export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
```

Modify your `$PATH` to include the location of the eHive scripts:

```
export PATH=${ehive_dir}/scripts/:${PATH}
```

- Install dependency
  1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder
  2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`
  3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

  And you are ready to go!
- Conventions used in this section
  - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
  - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a `ReseqTrack` database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                         # is the name you want
                                                                    ␣
↪                         # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::VCFilterGATK \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::VCFilterGATK.pm module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                 -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                 -logic_name find_files \
                 -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.7 VCF decoration

This workflow is used in IGSR for annotating/validating the VCF files produced in our project. More specifically the pipeline will take the phased VCF generated after running the PyHive::PipeConfig::Shapeit.pm pipeline and will add the following annotations:

- Allele frequency for a particular variant
- Alelle frequency in the different populations analyzed
- Total number of alternate alleles in called genotypes
- Total number of alleles in called genotypes
- Number of samples with data
- Information on whether a variant is within the exon pull down target boundaries
- Approximate read depth

Additionally, the workflow will also check if the VCF is in a valid format

2.7.1 Dependencies

- Nextflow

  This pipeline uses a workflow management system named Nextflow. This software can be downloaded from:

  https://www.nextflow.io/

- Tabix and bgzip

  Tabix and bgzip are part of the HTSlib project, which can be downloaded from:

  https://github.com/samtools/htslib

- BCFTools

  Downloadable from:

  http://www.htslib.org/download/

- BEDTools

  Downloadable from:

  https://github.com/arq5x/bedtools2/releases/

- vcf-validator

  This tool can be obtained from:

  https://github.com/EBIvariation/vcf-validator

- IGSR-analysis code base

The scripts needed to run this workflow can be downloaded by cloning the IGSR-analysis github repo from:

https://github.com/igsr/igsr_analysis.git

2.7.2 How to run the pipeline

- First, you need to create a `nexflow.config` file that can be used by Nextflow to set the required variables. Here goes an example of one of these files:

```
params.sample_panel='/homes/ernesto/lib/igsr_analysis_master/igsr_analysis/
↪SUPPORTING/integrated_allsamples.20180619.superpopulations.panel'
params.pops='EAS,EUR,AFR,AMR,SAS' // comma separated list of populations or␣
↪superpopulations that will be used for the annotation
params.exome='/nfs/production/reseq-info/work/ernesto/isgr/VARIANT_CALLING/
↪VARCALL_ALLGENOME_13022017/COMBINING/ANNOTATION/output_1000G_Exome.v1.ensembl.
↪bed' // path to .BED file with coordinates of the exomes
params.tabix='/nfs/production/reseq-info/work/ernesto/bin/anaconda3/bin/tabix' //␣
↪path to tabix binary
params.igsr_root='/nfs/production/reseq-info/work/ernesto/isgr/SCRATCH/17_09_2018/
↪lib/igsr_analysis/' // folder containing the igsr codebase downloaded from␣
↪https://github.com/igsr/igsr_analysis.git
params.vcf_validator='/nfs/production/reseq-info/work/ernesto/bin/vcf_validator/
↪vcf_validator_linux' // path to vcf_validator binary
params.bcftools_folder='~/bin/bcftools-1.6/' // folder containing the bcftools␣
↪binary
```

- Then, you can start your pipeline by doing:

```
nextflow -c nextflow.config run $IGSR_CODEBASE/scripts/VCF/ANNOTATION/decorate.nf␣
↪--phased_vcf chr20.unannotated.phased.vcf.gz --ann_vcf chr20.ann.unphased.vcf.
↪gz --region 20:1-64444167
```

Where:

- `-c` option allows you to specify the path to the `nextflow.config` file
- `$IGSR_CODEBASE` is the folder containing the igsr codebase downloaded from `https://github.com/igsr/igsr_analysis.git`
- `--phased_vcf` is the phased VCF generated after running the Py-Hive::PipeConfig::INTERGRATION::Shapeit.pm pipeline that will be decorated in this workflow. You will need to create a tabix index for this VCF
- `--ann_vcf` is the unphased VCF generated by the Py-Hive::PipeConfig::INTEGRATION::VCFIntegrationGATKUG.pm pipeline which will contain the 'INFO/DP' (depth) annotation for each particular site. You will need to create a tabix index for this VCF
- `--region` is the region that will be analyzed

2.7.3 Pipeline output

This worklow will create a folder name `results/` with 2 output files:

- `chr20.GRCh38.phased.vcf.gz`

  That will be the final annotated VCF

- `chr20.vcf.validation.txt`

  Will contain the output of the `vcf-validator`

2.8 ML-based workflow to filter a VCF

Filtering the spurious variants from a callset is a common task in variation studies using sequencing data.

Variant discovery methods are not perfect and will produce a certain number of false positive calls, specially if the sequencing data is either noisy or the depth of coverage is not enough to distinguish a real variant from a sequencing artifact.

This is why a method for identifying these false variants is necessary. Different methods have been developed for filtering and at the time of writing I would say that the most used is GATK VQSR which works really well and is specially relevant for filtering the calls obtained with the GATK callers (UnifiedGenotyper [UG] and HaplotypeCaller [HC]).

VQSR relies on a sophisticated model that needs to be trained with the annotation profiles generated by UG or HC for the different variant sites and will also depend on the existence of reference datasets specially formatted in order to be used with VQSR.

The problem arises when you need to filter a callset obtained from a non-GATK caller and do not have the variant annotations required by VQSR or you are analysing variation data from a non-human organism for which there is not a VQSR-formatted reference call set.

If you find yourself in this situation you might find this pipeline useful.

2.8.1 Foundation of the filtering

This pipeline implements a supervised Machine Learning (ML) model in order to solve a binary classification problem. It is sup
and it is a binary classification problem where we have multiple numerical independent variables (annotation values for each of the variant sites) to predict or classify a binary outcome (is a certain site a real variant?). This particular type

of problem can be modelled using a Logistic regression binary classifier and more specifically our pipeline uses the implementation from the Scikit-learn Python library

This pipeline needs to be run in different stages

1) Recursive Feature Elimination (RFE) stage (optional). This pipeline uses the Scikit-learn RFE implementation and it works by recursively removing features (annotations), building a logistic regression model using the remaining attributes and calculating the model accuracy. RFE is able to work out the combination of n attributes that contribute most to the prediction
2) Training the ML model for the SNPs and INDELs independently
3) Applying the fitted model generated in step 2 trained model on the VCF that you want to filter

- Recursive Feature Elimination

This step will

2.8.2 USAGE

This page is under construcion

2.9 Benchmarking with a true set

This workflow is used in our project to benchmark the sites identified in our call sets for a certain sample with a reference call set for which the calls have enough quality to be considered true positives.

Additionally, this workflow will also compare the phased genotypes in our call sets with the ones identified in the reference call set.

2.9.1 Dependencies

- Nextflow

  This pipeline uses a workflow management system named Nextflow. This software can be downloaded from:

  https://www.nextflow.io/

- Bgzip and Tabix

  Bgzip and Tabix are part of the HTSlib project, which can be downloaded from:

  https://github.com/samtools/htslib

- BCFTools

  Downloadable from:

  http://www.htslib.org/download/

- IGSR-analysis code base

  The scripts needed to run this workflow can be downloaded by cloning the IGSR-analysis github repo from:

  https://github.com/igsr/igsr_analysis.git

- vcflib

  This library can be downloaded from:

  https://github.com/vcflib/vcflib

Using Docker or Singularity

If you want to skip the installation of these dependencies you can pull the Docker Image from:

```
docker pull elowy01/igsr_analysis:latest
```

or build the Singularity image by using:

```
singularity build igsr_analysis.simg docker://elowy01/igsr_analysis
```

2.9.2 How to run the pipeline

- You can start your pipeline by doing:

```
nextflow run $IGSR_CODEBASE/scripts/VCF/QC/BENCHMARKING_TRUESET/vcf_eval.nf --vt␣
↪snps --vcf VCF --true TRUE.vcf.gz --filt_str "PASS,." --chros chr20
--high_conf_regions highconf.file.bed --calc_gtps true -with-singularity igsr_
↪analysis.simg
```

Where:

- `$IGSR_CODEBASE` is the folder containing the igsr codebase downloaded from `https://github.com/igsr/igsr_analysis.git`
- `--vcf` is the VCF that will be benchmarked with the true set. Notice that you will need to create a tabix index of this file before running this pipeline
- `--true` is the path to the VCF containing the true call set
- `--chros` is the chromosome that will be analyzed. i.e.: `chr20`
- `--vt` is the parameter used to set the type of variants that will be analyzed. i.e. `'snps'/'indels'`
- `--calc_gtps` if true, then the genotype concordance between `--true` and `--vcf` will be calculated
- `--filt_str` Filter string used by BCFTools in order to subset a certain subset of variants to be analysed. i.e. `"PASS,."`
- `--high_conf_regions` BED file used to control the genomic regions for which the benchmarking will be done. This parameter is optional
- `--calc_gtps` If 'true' then the genotype concordance between `--vcf` and `--true` will be calculated
- `-with-singularity` Parameter used to specify the Singularity image that Nextflow will use to run the workflow. This parameter is optional

2.9.3 Pipeline output

This worklow will create a folder name `results/` with the following relevant files:

- `TP_true.vcf.gz`

Will contain the set of sites that were idendified both in our call set and in the true call set

- `TP.stats`

Are the stats calculated by running `bcftools stats TP.vcf.gz`

- `FP.vcf.gz`

Will contain the set of sites identified in our call set and absent in the true call set

- `FP.stats`

Are the stats calculated by running `bcftools stats FP.vcf.gz`

- `FN.vcf.gz`

Will contain the set of sites that were not idendified in our call set and are present in the true call set

- `FN.stats`

Are the stats calculated by running `bcftools stats FN.vcf.gz`

- `TP_true.highconf.vcf.gz`

Will contain the set of sites from the true call set that were idendified both in our call set and in the true set but restricted to the regions passed with `params.high_conf_regions`

- `TP_target.highconf.vcf.gz`

Will contain the set of sites from the target call set that were idendified both in our call set and in the true call set but restricted to the regions passed with `params.high_conf_regions`

- `TP.highconf.stats`

Are the stats calculated for the sites identified both in the true and target call sets

- `FP.highconf.vcf.gz`

Will contain the set of sites identified in our call set and absent in the true call set but restricted to the regions passed with `params.high_conf_regions`

- `FP.highconf.stats`

Are the stats calculated by running `bcftools stats FP.highconf.vcf.gz`

- `FN.highconf.vcf.gz`

Will contain the set of sites that were not idendified in our call set and are present in the true call set but restricted to the regions passed with `params.high_conf_regions`

- `FN.highconf.stats`

Are the stats calculated by running `bcftools stats FN.highconf.vcf.gz`

- `GT_concordance.txt`

This file contains the tables produced after comparing the phased genotypes in our call set with the true call set

2.10 Consensus call set generation

This pipeline is used to generate a consensus biallelic call set generated after combining different call sets that may have been generated by different variant callers (i.e. BCFTools, GATK, Freebayes).

This pipeline will normalize the VCFs and will generate the union of the sites present in each call set. These sites will then be used by the Genome Analysis ToolKit (GATK) UnifiedGenotyper (UG) in the Genotype Given Alleles (GGA) mode together with the BAM files used in the variant calling to recall the sites from the union list. Finally, the Variant Score Recalibration (VQSR) procedure will be used for filtering these recalled sites.

In order to run this workflow we need to do the following:

1. Preparing the environment

   Modify your `$PYTHONPATH` to include the required libraries:

   ```
   export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
   ```

   Modify your `$PERL5LIB` to include the required libraries:

   ```
   export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
   ```

   Modify your `$PATH` to include the location of the eHive scripts:

   ```
   export PATH=${ehive_dir}/scripts/:${PATH}
   ```

   - Install dependency

1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder
2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`
3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

And you are ready to go!

- Conventions used in this section
  - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
  - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

2. Databases

The pipeline uses two databases. They may be on different servers or the same server.

2.1 The ReseqTrack database

The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

In order to create a `ReseqTrack` database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                    ␣
↪                         # is the name you want
                                                                    ␣
↪                         # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

2.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

3. Initialise the pipeline

The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::INTEGRATION::VCFIntegrationGATKUG \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::INTEGRATION::VCFIntegrationGATKUG.pm module file. You will probably want to override the defaults for many of these options so take a look.

4. Seeding the pipeline

   In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                -logic_name find_files \
                -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

   Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed. This file must exist in the ReseqTrack database

5. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

6. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

   Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

   While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.11 Phasing pipeline

This workflow is used to generate a phased VCF by using Beagle/Shapeit and emulates the analyses from the [phase 3] of the 1000 genomes project. It can be run after generating and integrated call set by using the PyHive::PipeConfig::PHASING.pm

This pipeline is designed to be run for SNPs or INDELs independently or for both variant types together in the same VCF. *Important:* This workflow can only analyze biallelic variants and it will crash if you try to analyze multiallelic sites.

1. Input preparation:

This pipeline will take as input the VCF file that contains either SNPs, INDELs or both types together. In order to generate a combined VCF containing both SNPs+INDELs from 1 SNP VCF + 1 INDEL VCF you can do the following:

```
bcftools concat input1.snps.vcf.gz input2.indels.vcf.gz -o combined.snps_indels.vcf.
↪gz -Oz -a
```

2. Preparing the environment

   Modify your `$PYTHONPATH` to include the required libraries:

   ```
   export PYTHONPATH=${ehive_dir}/wrappers/python3/:$PYTHONPATH
   ```

   Modify your `$PERL5LIB` to include the required libraries:

   ```
   export PERL5LIB=${ehive_dir}/modules/:${igsr_analysis_dir}/:${PERL5LIB}
   ```

   Modify your `$PATH` to include the location of the eHive scripts:

   ```
   export PATH=${ehive_dir}/scripts/:${PATH}
   ```

   - Install dependency

     1) Clone repo by doing `git clone https://github.com/igsr/igsr_analysis.git` in the desired folder

     2) `pip install ${igsr_analysis_dir}/dist/igsr_analysis-0.91.dev0.tar.gz`

     3) Modify `$PYTHONPATH` to add the folder where your pip installs the Python packages

     And you are ready to go!

   - Conventions used in this section
     - `${igsr_analysis_dir}` is the folder where you have cloned `https://github.com/igsr/igsr_analysis.git`
     - `${ehive_dir}` is the folder where you have cloned `https://github.com/Ensembl/ensembl-hive.git`

3. Databases

   The pipeline uses two databases. They may be on different servers or the same server.

   3.1 The ReseqTrack database

   The pipeline queries a `ReseqTrack` database to find the VCF that will be filtered by the pipeline. It will also add file metadata for the final filtered VCF.

   In order to create a `ReseqTrack` database use the following

commands:

```
mysql -h <hostname> -P <portnumber> -u <username> -p???? -e "create␣
↪database testreseqtrack" # where testreseqtrack
                                                                  ␣
↪                         # is the name you want
                                                                  ␣
↪                         # to give to the ReseqTrack DB
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/table.sql
mysql -h <hostname> -P <portnumber> -u <username> -p????␣
↪testreseqtrack < $RESEQTRACK/sql/views.sql
```

- Conventions used in this section:

  `$RESEQTRACK` is the folder where you have cloned `https://github.com/EMBL-EBI-GCA/reseqtrack.git`

3.2 The Hive database

This is database is used by the Hive code to manage the pipeline and job submission etc. The pipeline will be created automatically when you run the `init_pipeline.pl` script. Write access is needed to this database.

4. Initialise the pipeline

The pipeline is initialised with the hive script `init_pipeline.pl`. Here is an example of how to initialise a pipeline:

```
init_pipeline.pl PyHive::PipeConfig::INTEGRATION::PHASING \
                 -pipeline_url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/
↪hive_dbname \
                 -db testreseqtrack \
                 -pwd $DB_PASS \
                 -hive_force_init 1
```

The first argument is the the module that defines this pipeline. Then `-pipeline_url` controls the Hive database connection details, in this example:

```
g1krw= username
$DB_PASS= password
mysql-rs-1kg-prod= hostname
4175= Port number
hive_dbname= Hive DB name
```

Then `-db` is the name of the Reseqtrack database name used in the section 2.1 `-pwd` is the ReseqTrack DB password

The rest of the options are documented in the PyHive::PipeConfig::INTEGRATION::PHASING.pm module file. You will probably want to override the defaults for many of these options so take a look.

5. Seeding the pipeline

In order to seed the pipeline with the VCF file that will be analyzed use the hive script `seed_pipeline.pl`:

```
seed_pipeline.pl \
                 -url mysql://g1krw:$DB_PASS@mysql-rs-1kg-prod:4175/hive_
↪dbname \
                 -logic_name find_files \
                 -input_id "{ 'file' => '/path/to/file/input_file.txt' }"
```

Where `-url` controls the Hive database connection details and `/path/to/file/input_file.txt` contains the filename of the VCF to be analyzed.

6. Sync the hive database

   This should always be done before [re]starting a pipeline:

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪sync
```

   where `-url` are the details of your hive database. Look at the output from `init_pipeline.pl` to see what your url is.

7. Run the pipeline

   Run e.g.:

```
beekeeper.pl -url mysql://g1krw:{password}@mysql-g1k:4175/my_hive_db_name -
↪loop &
```

   Note the '&' makes it run in the background.

   Look at the pod for `beekeeper.pl` to see the various options. E.g. you might want to use the `-hive_log_dir` flag so that all `output/error` gets recorded in files.

   While the pipeline is running, you can check the 'progress' view of the hive database to see the current status. If a job has failed, check the msg view.

2.12 CRAM to BAM conversion

This workflow is used in IGSR for:

1) Downloading a CRAM file from the archive (ENA, IGSR FTP, etc...)
2) Convert it to BAM
3) Create an index for converted BAM.

This workflow relies on the ASPERA service for the fast download of the data from the archives

2.12.1 Dependencies

- Nextflow

  This pipeline uses a workflow management system named Nextflow. This software can be downloaded from:

  https://www.nextflow.io/

- SAMTools

  Downloadable from:

  http://www.htslib.org/download/

- Aspera connect software:

  This ascp client can be obtained from:

  http://asperasoft.com/software/transfer-clients/connect-web-browser-plug-in/

2.12.2 How to run the pipeline

- First, you need to create a `nexflow.config` file that can be used by Nextflow to set the required variables. Here goes an example of one of these files:

```
params.samtools_folder='~/bin/samtools-1.9/' // folder containin the samtools␣
↪binary
// params defaults for ascp client
params.key_file = '/homes/ernesto/.aspera/connect/etc/asperaweb_id_dsa.openssh' //
↪ Private-key file name (id_rsa) for authentication
params.transfer_rate = '900M'
params.port = 33001 // TCP port used for SSH authentication
```

- Then, you can start your pipeline by doing:

```
nextflow -C nextflow.config run $IGSR_CODEBASE/scripts/FILE/cram2bam.nf --file␣
↪input.txt
```

Where:

- `-C` option allows you to specify the path to the `nextflow.config` file
- `$IGSR_CODEBASE` is the folder containing the igsr codebase downloaded from `https://github.com/igsr/igsr_analysis.git`
- `--file` File with the urls pointing to the CRAM files to be converted. This file shold have a content similar to:

```
url,dest,prefix
era-fasp@fasp.sra.ebi.ac.uk:/vol1/ERZ454/ERZ454001/ERR1457180.cram,/
↪path/in/dest/ERR1457180.cram,ERR1457180
```

Where `url` points to the location of the file to be downloaded, `dest` is the path in the local machine where it will be downloaded and `prefix` is used as the string used in the converted BAM file and its respective index

2.12.3 Pipeline output

This worklow will create a folder name `converted/` with 2 output files:

- `prefix.bam`

  BAM file resulting after converting the downloaded CRAM file

- `prefix.bam.bai`

  The index created after running `samtools index prefix.bam`

CHAPTER 3

Auto-documentation

Access the different modules and functions used in for the analysis of IGSR data

Contents:

3.1 VCF

Modules used to perform different operations on files in the VCF format

Contents:

3.1.1 VcfNormalize

3.1.2 VcfQC

3.1.3 VcfUtils

3.1.4 VCFfilter

Modules used to filter a VCF of spurious calls

Contents:

BCFTools

GATK

3.1.5 VCFIntegration

Modules to perform different operations aimed to integrate different call sets into a single consensus call set

Contents:

Shapeit

Beagle

SNPTools

3.2 BEDTools

Contents:

Created on 24 Apr 2017

@author: ernesto

class BEDTools.BEDTools.**BEDTools**(*bedtools_folder*)

Class used to perform different operations with the BEDTools package.

This is essentially a wrapper for the BEDTools package. The functionality is quite limited and additional functions will be added as necessary

Methods

make_windows(w, g[, s, subtract, lextend, ...])	This method will make windows from a genome file by using 'bedtools makewindows'

make_windows(*w*, *g*, *s=None*, *subtract=None*, *lextend=None*, *rextend=None*, *verbose=False*)

This method will make windows from a genome file by using 'bedtools makewindows'

Parameters

w [int] width of windows in bp

g [filename] Path to genome file

s [int , optional]

overlap in bp. i.e. if -w 100 -s 80 will generate:

chr1 0 100 chr1 80 180 chr1 160 260 ... So, -s defines the offset in bp

Another example -w 1000 -s 200

chr1 0 1000 chr1 200 1200 chr1 400 1400 chr1 600 1600

lextend [int, optional] Extend each interval to the left by int bases

rextend [int, optional] Extend each interval to the right by int bases

subtract [filename, optional] BED file containing the features that will be removed from the generated windows. For example, if we have the following window:

chr20 1000 2000

And we have the following feature in the BED file: chr20 1100 1200 Then the resulting windows will be like:

chr20 1000 1100 chr20 1200 2000

verbose [bool, optional] Default=False

Returns

——

list A list of lists. Each sublist is composed of ['chr','start','end'] It will return an empty list if not elements for a certain chr are defined

3.3 BamQC

Contents:

Created on 12 Oct 2016

@author: ernesto

class BamQC.BamQC.**BamQC**(*bam*, *samtools_folder=None*, *java_folder=None*, *picard_folder=None*, *chk_indel_folder=None*, *verifybamid_folder=None*)

Class to do the quality assessment on a BAM format file

Methods

aggregate_stats(cov_list)	Used to calculate aggregated stats on a list of SDepth objects
get_contigs()	Get all contigs from this BAM
get_simple_stats()	Get a dict with stats on the BAM file as calculated by samtools flagstat
list_of_readgroups()	Get the Read Groups extracted from the header of the BAM file
list_of_samples()	Get the samples names from the header of the BAM file
run_CollectHsMetrics(baits_file[, outfile, ...])	Run Picard's CollectHsMetrics on a Exome sequencing BAM file
run_CollectWgsMetrics(reference[, outfile, ...])	Run Picard's CollectWgsMetrics on a WGS BAM file
run_chk_indel_rg([outfile])	Run Heng Li's chk_indel_rg on a BAM file
run_samtools_depth(chros)	Calculate several coverage metrics on a whole genome sequencing BAM file using 'samtools depth'
run_verifybamid(genotype_file, outprefix[, ...])	Run VerifyBAMID to check for sample swap or contamination issues

aggregate_stats(*cov_list*)

Used to calculate aggregated stats on a list of SDepth objects

Parameters

cov_list [list] List containing the SDepth objects for which the stats will be aggregated.

Returns

A SDepth object

get_contigs()

Get all contigs from this BAM

Parameters

None

Returns

dict A dictionary containing the following information:

{'contig Name': length (in bp)}

get_simple_stats()
Get a dict with stats on the BAM file as calculated by samtools flagstat

Parameters

None

Returns

dict A dictionary containing the following information: {

"total_no_reads": int "no_duplicates": int "total_no_mapped": int "no_properly_paired": int }

list_of_readgroups()
Get the Read Groups extracted from the header of the BAM file

Parameters

None

Returns

list List composed of the read groups

list_of_samples()
Get the samples names from the header of the BAM file

Parameters

None

Returns

list List with the sample names

run_CollectHsMetrics(*baits_file*, *outfile=None*, *cov_cap=None*)
Run Picard's CollectHsMetrics on a Exome sequencing BAM file

Parameters

baits_file [filename] Path to the file containing the Exome baits.

outfile [filename, optional] If provided, then create a file with the output of this program

cov_cap [int, optional] Picard's Coverage Cap parameter. Treat positions with coverage exceeding this value as if they had coverage at this value. Default value: 250.

Returns

——

A CMetrics object

run_CollectWgsMetrics(*reference*, *outfile=None*, *cov_cap=None*)
Run Picard's CollectWgsMetrics on a WGS BAM file

Parameters

reference [filename] Fasta file used as the genome reference

outfile [filename, optional] If provided, then create a file with the output of this program

cov_cap [int, optional] Picard's Coverage Cap parameter. Treat positions with coverage exceeding this value as if they had coverage at this value. Default value: 250.

Returns

——

A CMetrics object

run_chk_indel_rg(*outfile=None*)
Run Heng Li's chk_indel_rg on a BAM file

Parameters

outfile [filename, optional] If provided, then create a file with the output of this program

Returns

list A list of Chk_indel objects

run_samtools_depth(*chros*)
Calculate several coverage metrics on a whole genome sequencing BAM file using 'samtools depth'

Parameters

chros [list or string] List of contigs or just a single contig used for calculating the coverage

Returns

——

List of SDepth objects

This method runs samtools depth on a BAM file and will calculate the following metrics:

- Number of Bases mapped: This is the number of bases having at least one read mapped
- Sum of depths of coverage: This is the sum of all the depths in each of the Bases mapped
- Breadth of coverage: This is the result of dividing bases_mapped/length(contig) (i.e. what portion of the contig has reads mapped)
- Depth of coverage: This is the result of dividing sum_of_depths/length(contig)

run_verifybamid(*genotype_file*, *outprefix*, *outdir=None*)
Run VerifyBAMID to check for sample swap or contamination issues

Parameters

genotype_file [filename] vcf file with chip genotypes to use

outprefix [str] prefix for outputfiles

outdir [str, optional] If provided, then put output files in this folder

Returns

list A list with the paths to the output files generated by VerifyBAMID

class BamQC.BamQC.**CMetrics**(*metrics*, *cov_data*)
Class to store coverage information on the metrics calculated by Picard's CollectHsMetrics/CollectWgsMetrics on an Exome or WGS BAM file

Methods

create_cov_barplot(filename[, xlim, ylim])	This method will create a Barplot using the different coverage values counts calculated by Picard's CollectHsMetrics or CollectWgsMetrics
print_report([filename])	Used to print a text report of data in the object

create_cov_barplot (*filename*, *xlim=None*, *ylim=None*)
This method will create a Barplot using the different coverage values counts calculated by Picard's CollectHsMetrics or CollectWgsMetrics

Parameters

filename [filename] PDF file to write the plot.

xlim [tuple, optional] Set the X-axis limit

ylim [tuple, optional] Set the Y-axis limit

print_report (*filename=None*)
Used to print a text report of data in the object

Parameters

filename [filename, optional] Filename to write the report. The default is STDOUT.

class BamQC.BamQC.**Chk_indel** (*RG*, *ins_in_short_homopolymer*, *del_in_short*, *ins_in_long*, *del_in_long*, *outcome=None*)
Class to store information on the ratio of short insertion and deletion calculated by runnint Heng Li's chk_indel_rg

Methods

calc_ratio()	Method to calc ratio ins-in-short-homopolymer/del-in-short and check if it is > 5

calc_ratio ()
Method to calc ratio ins-in-short-homopolymer/del-in-short and check if it is > 5

Returns

str It returns PASS/FAILED depending on the outcome of the test

class BamQC.BamQC.**SDepth** (*contig=None*, *mapped=None*, *breadth=None*, *depth=None*, *length=None*, *sum_of_depths=None*, *max=None*)
Class to store coverage metrics on a Whole Genome Sequencing BAM file calculated using SAMtools depth

Methods

print_report([filename])	Used to print a text report of data in the object

print_report (*filename=None*)
Used to print a text report of data in the object

Parameters

filename [filename, optional] Filename used to write the report. The default is STDOUT.

3.4 VariantCalling

Modules used to identify variants from a file in the BAM format Here you can find wrappers for GATK UnifiedGenotyper and HaplotypeCaller

Contents:

3.4.1 GATK

3.5 Utils

Utils contains a series of modules used in different aspects of the project

Contents:

3.5.1 RunProgram

Python Module Index

b

BamQC.BamQC, 35
BEDTools.BEDTools, 34

Index

A

aggregate_stats() (BamQC.BamQC.BamQC method), 35

B

BamQC (class in BamQC.BamQC), 35
BamQC.BamQC (module), 35
BEDTools (class in BEDTools.BEDTools), 34
BEDTools.BEDTools (module), 34

C

calc_ratio() (BamQC.BamQC.Chk_indel method), 38
Chk_indel (class in BamQC.BamQC), 38
CMetrics (class in BamQC.BamQC), 37
create_cov_barplot() (BamQC.BamQC.CMetrics method), 38

G

get_contigs() (BamQC.BamQC.BamQC method), 35
get_simple_stats() (BamQC.BamQC.BamQC method), 36

L

list_of_readgroups() (BamQC.BamQC.BamQC method), 36
list_of_samples() (BamQC.BamQC.BamQC method), 36

M

make_windows() (BEDTools.BEDTools.BEDTools method), 34

P

print_report() (BamQC.BamQC.CMetrics method), 38
print_report() (BamQC.BamQC.SDepth method), 38

R

run_chk_indel_rg() (BamQC.BamQC.BamQC method), 37
run_CollectHsMetrics() (BamQC.BamQC.BamQC method), 36
run_CollectWgsMetrics() (BamQC.BamQC.BamQC method), 36
run_samtools_depth() (BamQC.BamQC.BamQC method), 37
run_verifybamid() (BamQC.BamQC.BamQC method), 37

S

SDepth (class in BamQC.BamQC), 38