GC NGS PIPELINE

Release 1.0a1

Bernat del Olmo, Xavier Sumpsi, Carles Ferrer-Costa

Jul 24, 2022

CONTENTS:

Navigator

CHAPTER

ONE

GC NGS PIPELINE

1.1 Introduction

GC NGS PIPELINE is a command-line tool to handle DNA-seq analysis from Gencardio Diagnostics (GenDx).

This software should follow these non-functional requirements:

1. **Modular**: each step must be able to be executed independently throught sub-commands.
2. **Performant**: the system must be able to process sequencing data and store all relevant information fast.
3. **Lab interaction**: the system must interact with other applications and databases from the lab.
4. **Data integrity**: the system should check and validate the user inputs.
5. **Maturity**: the system has to run tests every time that a change is made, and new tests have to be built for new features. Both unit and inter-module tests should be done.
6. **Changeability**: everything is very likely to change, so the system must be able to handle any kind of changes in features. Configuration is key here.

Note: This software has been developed and tested with Python3.6 under a Linux Mint 20 server.

1.2 Scope

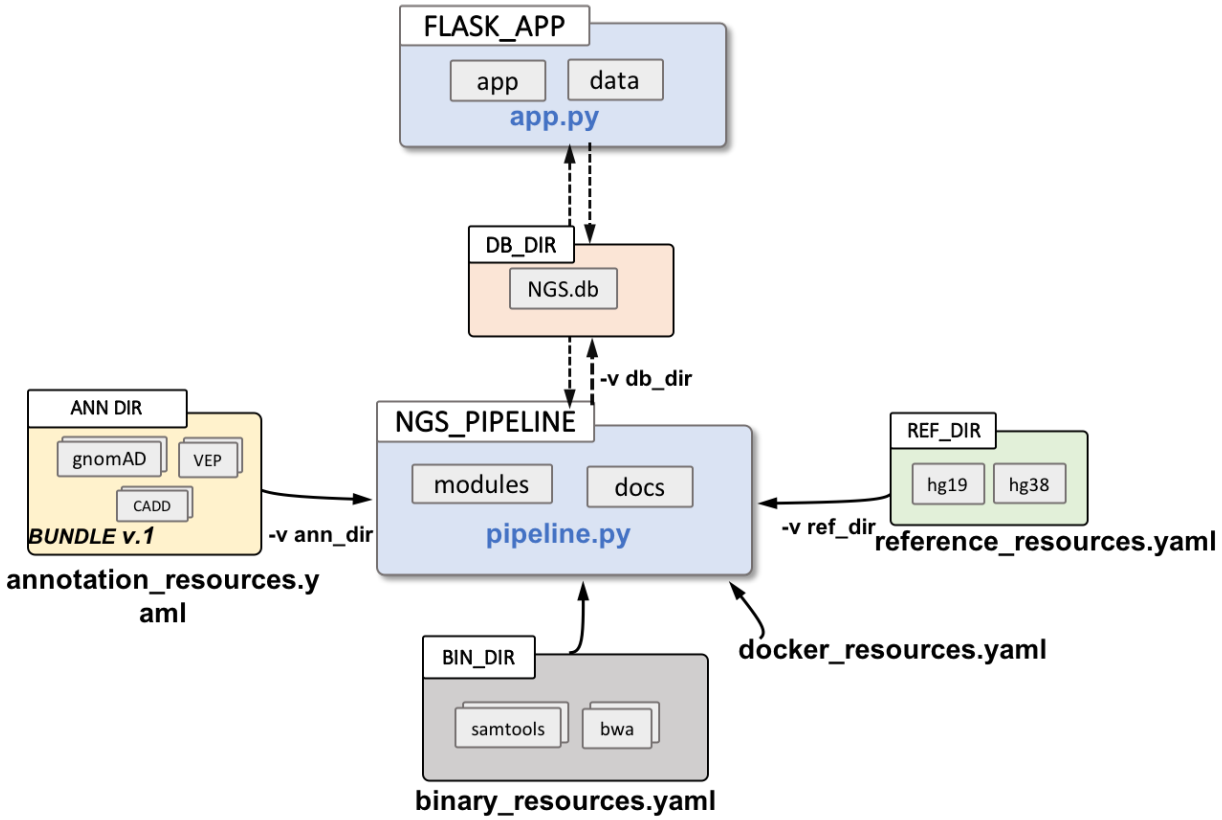
This software is intended to cover these NGS applications:

- Gene panel analysis:
  - Germline variants
  - Somatic variants
- Low-pass CNV detection.
- Exome sequencing analysis.
- Genome sequencing analysis.

The main steps of this pipeline can be executed independently (e.g. annotate an input VCF) or in its full extent (see `all` option)

1.3 Lab integration

NGS_PIPELINE is being developed to interact with other services from our lab. This picture illustrates a first integration scheme with a `flask app` by sharing a common database structure that is used and fed bidirectionally.



1.4 Resource configuration:

Long-term development and maintainance of genomic pipelines is non-trivial since software and genomic annotation resources are evolving constanly. Therefore, we need a system that allows to rapidly configure new resources of modify existent versions. One way to keep track of this changes is by using YAML config files.

There are currently four external configuration files:

- `binary_resources.yaml` : software binaries/scripts and its versions.

Here a few lines:

```
version: 1.0
bin_dir: /path/to/bin_directory
samtools:
  binary: samtools
  version: 1.7
```

- `reference_resources.yaml`: genome reference files (fasta, dicts, gene lists, chrom sizes).

Here a few lines:

```
version: 1.0
ref_dir: /path/to/ref_directory
hg19:
  dirname: ""
  fasta: ucsc.hg19.fasta
  dict: ucsc.hg19.dict
  gene_bed: genelist.hg19.bed.gz
  chrom_sizes: hg19.txt
```

- `annotation_resources.yaml`: genomic annotations (bed, vcf, bigWig)

Here a few lines:

```
version: 1.0
ann_dir: /path/to/ann_directory
resources:
  gnomAD:
    version: "2.1.1"
    resource_name: gnomAD
    hg19:
      items:
        -1:
          key: GNOMAD_FILE
          file: gnomad.genomes.r2.1.1.sites.only_af.vcf.gz
        -2:
          key: GNOMAD_ONLY_AF_FILE
          file: somatic-b37_af-only-gnomad.raw.sites.chr.vcf.gz
```

- `docker_resources.yaml`: docker images

Here a few lines:

```
gatk:
  image: broadinstitute/gatk:4.1.3.0
  version: 4.1.3.0
picard:
  image: broadinstitute/picard
  version: ""
```

1.5 Sub-commands:

By prompting `python3 gc_ngs_pipeline.py` you will have access to the following sub-commands:

The usage information from each subcommand can be shown by with `-h` and `--help` options.

all Run all steps (map, call, annotate, report) successively

```
# Start from raw FASTQ files, run all NGS pipeline (map, call, annotate, report)
python3 gc_ngs_pipeline all -s <targeted,wgs,lowpass> \
  -r <hg19/hg38> -t <num_cpus> \
  -i input_fastq_dir/ -o output_dir/ \
  --lab_data <lab_xlsx> \
  --db_dir database_dir/ \
```

(continues on next page)

(continued from previous page)

```
  --user_id <user_id> \
  --ann_yaml annotation_resources.yaml \
  --docker_yaml docker_resources.yaml \
  --ref_yaml reference_resources.yaml \
  --bin_yaml binary_resources.yaml
```

map Preprocess and map fastq files

```
# Map FASTQ files
python3 gc_ngs_pipeline map -s <targeted,wgs,lowpass> \
  -r <hg19/hg38> -t <num_cpus> \
  -i input_fastq_dir/ -o output_dir/ \
  --lab_data <lab_xlsx> \
  --db_dir database_dir/ \
  --user_id <user_id> \
  --ann_yaml annotation_resources.yaml \
  --docker_yaml docker_resources.yaml  \
  --ref_yaml reference_resources.yaml \
  --bin_yaml binary_resources.yaml
```

call Call somatic/germline variants

annotate

```
Enrich variants using various genome annotations
```

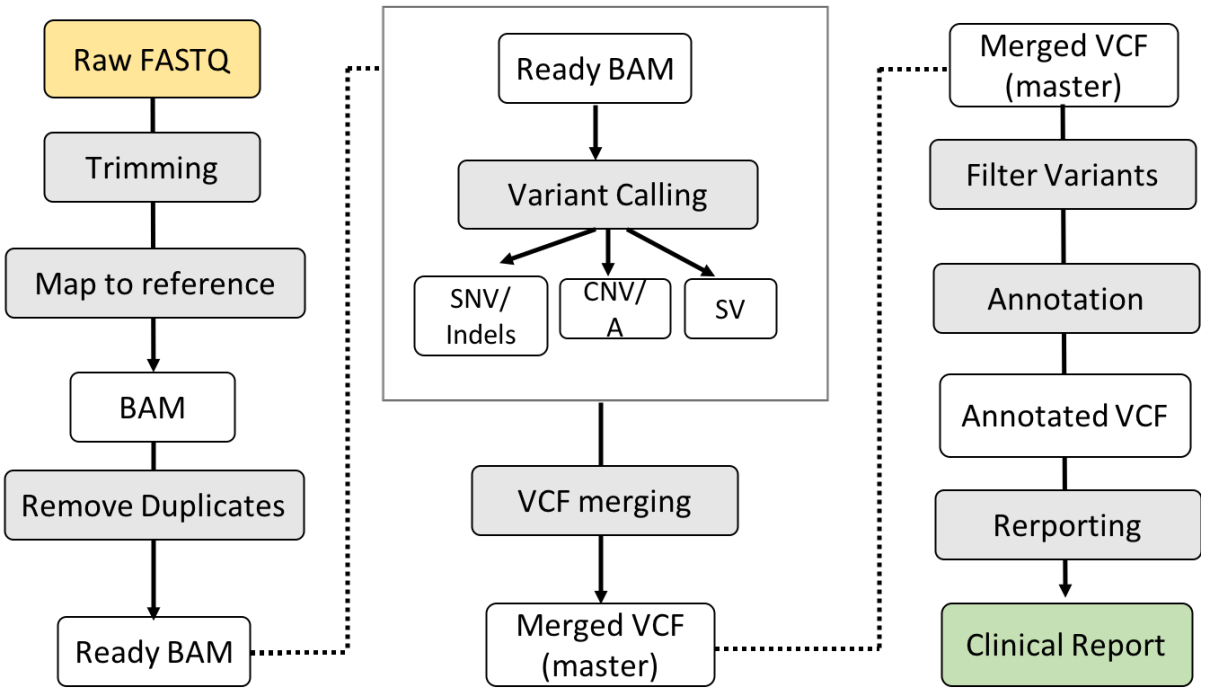
report NGS QC reports

```
TODO
```

sarscov2 DEPRECATED. Sars-cov-2 analysis was added as an *ad-hoc* sub-command for convenience. Complete analysis starting from raw fastqs and ending with lineage classification

1.6 Pipeline description:

This part descripes the main workflow for analyzing short-read sequences.



1.6.1 Pre-processing

- **Trimming**:
  - **Low quality bases** towards 5' ends are often found on 2nd Gen NGS sequencers. Trimming them is needed specially when reads get longer than 76bp (google this: "300bp MiSeq" and check FastQC base-quality histogram)
  - **Sequencing adapters** are short oligos that enable multiple samples to be sequenced together (using index sequences) and are also identified for the sequencer as valid fragments (P5, P7 sequences). These sequencers may be present at the raw FASTQ files whe the insert is smaller than the read length itself. Trimming them may also be convenient for downstream analysis
- **UMI processing**: Unique Molecular Identifiers (UMIs) allow to identify barcodes are short sequences used to uniquely tag each molecule in a sample library. UMIs are located within the adapter sequence.

The current software version uses fastp since it is widely used and it provides a solution for all the aforementioned requirements.

1.6.2 Mapping short reads to a genome reference

Short-read mapping/alignment is the core process of any NGS pipeline. This is because the sequencer produces FASTQ files without genomic context. Thus, we first need to locate them and assign a confidence (mapping quality) accordingly.

This process takes FASTQ files as input and produces Sequence Alignment Map (SAM) or its binary counterpart (BAM). The SAM/BAM format was stablished as an standard to handle alignment meta data.

The current software version uses `BWA-MEM` (Maximum-Exact Matches) as its main mapper due to its widespread usage and capabilities.

1.6.3 Removing PCR/Optical read duplicates

When preparing a DNA library for being sequenced, it is often desired to perform a PCR amplification once adapters have been ligated to ensure that enough DNA molecules are available. Thus, multiple copies of the original genomic DNA molecule are intentionally created producing some degree of PCR duplicates.

PCR polymerases have a low probability to introduce errors. However, sequencing errors in the first cycles of amplification will be magnified at the next steps. So, after sequencing these errors may be confounded as true variants. It is recommended to remove PCR duplicates on non-amplicon sequencing.

The current software version uses `Picard MarkDuplicates` for this purpose.

1.6.4 Germline SNV/Indel variant calling

While SNV calling is gererally a solved problem (only on accessible regions), d etecting short INDELs (1-20 bp) imposes additional challenges. INDELs tend to be accumulated on repeats, which are often enriched in sequencing errors. Some studies have revealed that erroneous INDELs are common in homopolymer A/T regions. Additionally, INDEL detection is more difficult in WES due to inherent capture biases.

Calling variants in germline mutations is quite different than calling somatic mutations. Germline mutations are expected to have a 50% or 100% allele frequencies; hence, variant calling is mainly aimed to choose the most likely genotype between AA, AB or BB.

Latests improvements on the field have been the incorporation of local assembly (GATK HaplotypeCaller), and the use of bayesian haplotying strategies (Freebayes, Octopus).

The current software version uses a combination of `GATK HaploTypeCaller` and `Octopus`.

1.6.5 Somatic SNV/Indel variant calling

Analyzing variants from tumor specimens has many challenges compared to germline variant canalysis:

- Over time, FFPE-dervied DNA may be downgraded in quality by artifacts such as deamination (C>T transitions).
- Tumour purity (proportion of tumoral cells with respect to the total cells) heavily influences the observed variant allele frequency (VAF).
- Subclonallity, clones at different proportions.
- Copy number aberrations and Loss of Heterozigosity (LOH).
- A low DNA input that is translated in a high degree of PCR duplicates.

Some sophisticated approaches have been developed to overcome these limitations:

- Mutect2 uses a probabilistic method to determine wheter a variant is somatic, by comparint a tumor sample with a Panel of Normals (PoN)
- Lancet uses and enhanced de bruijn local assembly procedure to improve indel detection.

UMIs can be used to reduce DNA polymerase errors by consensus base calling. The current software version uses a combination of `GATK Mutect2` for SNVs and `Lancet` for INDELs.

1.6.6 Germline CNV detection on targeted sequencing

CNV detection in targeted sequencing data (WES, gene panels) is challenging as the sparse nature of its sequencing usually is translated in the absence of breakpoint signatures. Consequently, most callers have been designed to detect CNVs using only RD signals, with few exceptions. Even when depth of coverage is higher in WES/targeted sequencing applications, there are much more fluctuations due to inconsistent capture efficiency among targeted regions.

- Detecting CNVs on clinical gene panels:

  compared to Exomes, gene panel sequencing offers a higher read depth. This is translated in a higher resolution (up to a single-exon) for CNV analysis. There are currently a few tools that can be used for that purpose: DECoN, GRAPES (from my thesis), CoNVaDiNG, panelcn.MOPS are good examples.

  However, CNV analysis on gene panels is in general difficult (lots of false-positives), It is recommended to filter calls using visual inspection and have an updated record of previous results to discard possible false positives.

- Detecting CNVs on whole-exome:

  Whole-exome sequencing offers the possibility to detect CNVs typically larger than 10kb (multi-exon CNVs). Trying to detect single-exon CNVs may be not possible, since a lower coverage may introduce more fluctuations.

1.6.7 Somatic CNA (SCNA) detection on targeted sequencing

Similarly than with germline CNV detection, the available algorithm use relative read depth to derive CN status from a tumoral sample. Nonetheless, the cancer genome is more complex, and read depth alone is not enough to have an derive an accurate genome picture. Other signals, such as copy-neutral loss of heterozygosity (CN-LOH), tumour admixture and potential contamination must be considered.

Ideally, with no budget limitations, the best experimental design would consider the sequencing of tumour-normal pairs. Alternatively, it is recommended to sequence a panel of normal samples (PoN).

Available tools that support PoN are: GATK4 and CNVKit.

The current software version uses a combination of `CNVkit` and a PoN of 10 FFPE samples.

1.6.8 Structural Variant (SV) analysis

In the past few years, the widespread usage of paired-end sequencing data has enabled the exploration of genomic SVs at a much finer extent than before. A key contribution can be attributed to the continuous development of more specialized algorithms, capable of taking advantage of sequencing improvements (e.g. read length). As a consequence, now we have a much accurate picture of SV size distribution and SV subtypes.

Best results are obtained from integrative callers that can use multiple signals (Read-Pairs, Split-Read, Read Depth and Assembly) Such a stepwise approach can result in an increase of specificity, but it cannot improve the number of true positives. Some integrative callers such as Lumpy, GRIDSS, and Manta can call SVs without requiring a pre-specified signal, so they are able to increase the overall sensitivity rate.

In Cancer, SVs are of spencial interest due to the existatnce of targeted therapies for some gene fusions such as ALK-EML4. Gene fusions are created by SVs.

The current software version uses a combination of `Manta` and `Lumpy`.

1.6.9 Annotation resources

The Variant Call Format (VCF) has become the de facto format for describing variants. This format was originally created by the 1000 Genome Project because it was needed to stablish a common format to annotate genomic locus. For a comprehesive description check: Annovar description

Integrating rich genomic annotations within VCFs is essential for downstream biological/clinical analysis. Some existing annotation engines such as Variant Effect Predictor (VEP), SnpEff or ANNOVAR provide gene-based annotation that can be leveraged afterwards.

This pipeline uses VEP extensively by either using its pre-defined annotation fields or by setting custom annotations (vcf, bed, bigWig) The available annotation resources are specifice in the configuration file: `annotation_resources.yaml`

Find here a resource list that come with the `annotation bundle`:

Resource	Version
VEP	101
gnomAD	2.1.1
1000Genomes	20130502
dbNSFP	4.1
dbscSNV	1.1
spliceAI	.
Blacklist	.
Phastcons	100way
PhyloP	100way
mappability	100mer
CGI	1
ncER	.
chimerKB	4

1.6.10 NGS QC reports

NGS analysis on the clinical field requires the delivery of custom-made genetic reports. We used historically JaspeReports (v6.15) as a background reporter. Today we make use of high quality html to pdf renderers such as WeasyPrint that enables to keep developing within the same python environment.

CHAPTER

TWO

MODULES

2.1 annotate module

annotate.**annotate_cancer_hotspots**(*sample_list*, *ann_conf*)

Annotate cancer hotspots: Single residue and in-frame indel mutation hotspots identified in 24,592 tumor samples by the algorithm described in [Chang et al. 2017] and [Chang et al. 2016]

Parameters

- **sample_list** (*list*) – list of sample objects
- **bin_conf** – BinaryConfig object
- **ann_conf** – AnnotationConfig object

Returns list of sample objects

Return type list

annotate.**annotate_flanking_genes_to_sv**(*sample_list*, *bin_conf*, *ref_conf*)

This function appends flanking genes from SV calls.

Parameters

- **sample_list** (*list*) – list of sample objects
- **bin_conf** – BinaryConfig object
- **ref_conf** – GenomeConfig object

Returns list of sample objects

Return type list

annotate.**annotate_known_fusions**(*sample_list*, *bin_conf*, *ann_conf*)

Annotate SVs using chimerKB known fusions. Create a new annotated VCF and json

Parameters

- **sample_list** (*list*) – list of sample objects
- **bin_conf** – BinaryConfig object
- **ann_conf** – AnnotationConfig object

Returns list of sample objects

Return type list

annotate.**annotate_overlapping_genes_over_cnas**(*sample_list*)

annotate.**annotate_sv_with_vep**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *ann_conf*, *docker_conf*)

Annotate raw VCF files with Ensembl's Variant Effect Predictor (VEP). Create a new annotated vcf and json.

Parameters

- **sample_list** (*list*) – list of sample objects
- **analysis_conf** – AnalysisConfig object
- **bin_conf** – BinaryConfig object
- **ref_conf** – GenomeConfig object
- **ann_conf** – AnnotationConfig object
- **docker_conf** – DockerConfig object

Returns list of sample objects

Return type list

annotate.**annotate_with_vep**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *ann_conf*, *docker_conf*)

Annotate raw VCF files with Ensembl's Variant Effect Predictor (VEP). Create a new annotated vcf and json.

Parameters

- **sample_list** (*list*) – list of sample objects
- **analysis_conf** – AnalysisConfig object
- **bin_conf** – BinaryConfig object
- **ref_conf** – GenomeConfig object
- **ann_conf** – AnnotationConfig object
- **docker_conf** – DockerConfig object

Returns list of sample objects

Return type list

annotate.**classify_in_somatic_tiers**(*sample_list*, *bin_conf*, *ann_conf*)

Classification based on CIViC of somatic variants in three categories: 1. Drug sensitive variants 2. High impact variants with no CIViC evidence records (stop codons,

frameshifts, missense variants with high REVEL scores, splicing variants)

3. Other rare variants (intronic, intergenic, synonymous)

Parameters

- **sample_list** (*list*) – list of sample objects
- **bin_conf** – BinaryConfig object
- **ann_conf** – AnnotationConfig object

Returns list of sample objects

Return type list

annotate.**launch_annotation**(*args*, *sample_list*)

Main annotation function

Parameters

- **args** – pipeline command line arguments
- **sample_list** (*list*) – list of sample objects

Returns list of sample objects

Return type list

annotate.**merge_cnv_sv**(*sample_list*, *bin_conf*)

annotate.**merge_vcfs**(*sample_list*, *bin_conf*)

Merge VCFs (snv, CNV and SV) with bcftools. Create a new annotated vcf and json. :param list sample_list: list of sample objects :param bin_conf: BinaryConfig object :returns: list of sample objects :rtype: list

annotate.**select_analysis_genes_and_isoforms**(*sample_list*, *analysis_conf*)

Output a new VCF with a single isoform as follows:

- Dump a configured isoform if available through Gene Panel API
- Otherwise return canonical transcript from Ensembl

Parameters **sample_list** (*list*) – list of sample objects

Returns list of sample objects

Return type list

2.2 bam module

class bam.**Bam**(*bam*, *bin_config*, *docker_config*)

Bases: object

Class to handle Bam operations such as bam indexing, pcr duplicate removal, and qc metrics extraction.

Class instantiation automatically checks for bam consistency, header and read group availability

Parameters

- **bam** (*str*) – bam file
- **bin_config** – binary configuration object
- **docker_config** – docker configuration object

quick_check() *index()* *remove_duplicates()* *get_perc_duplicates()* *get_total_reads()* *get_read_length()* *get_mean_isize()* *get_sd_isize()* *get_coverage_metrics()*

property contigs: list

Getter returns a list of contigs

Variables **_header** (*str*) – bam header

Return type list(dict)

get_coverage_metrics(*panel_name=None*, *bed=None*, *output_dir=None*)

Calculate coverage metrics with Mosdepth

Parameters

- **bed** (*str*) – optional bed file when dealing with targeted sequencing applications
- **ouput_dir** (*str*) – output directory to write coverage files

Returns mosdepth coverage file, coverage dictionary

Return type str

Return type dict

get_mapping_metrics()

get_mean_isize(*N=5000*)

Get the mean insert size

Parameters **N** (*int*) – use up to N alignments to compute the mean insert size in basepairs

Variables **_bin_config.samtools.binary** (*str*) – samtools binary

Returns mean insert size

Return type float

get_perc_duplicates()

Return the percentage of PCR/Optical duplicates :var str _sample_name: sample name :var str _picard_metrics_file: metrics file created using picard :raises FileNotFoundError: when picard_metrics_file is not found :rtype: float

get_read_length(*N=5000*)

Get the mean read length

Parameters **N** (*int*) – use up to N alignments to compute the mena read length

Variables **_bin_config.samtools.binary** (*str*) – samtools binary

Returns mean read length

Return type int

get_sd_isize(*N=5000*)

Get the insert size standard deviation

Parameters **N** (*int*) – use up to N alignments to compute the std. deviation insert size. Default N = 5000

Variables **_bin_config.samtools.binary** (*str*) – samtools binary

Raises **ValueError** – N must be an integer value > 0

Returns insert size std. deviation

Return type float

get_total_reads(*bed=None*)

Get the total number of reads

Parameters **bed** (*str*) – optional bed file to restrict calculation over ROIs

Variables **_sample_name** (*str*) – sample name

Returns total number of reads

Return type int

property header: str

Getter return header from bam file

Variables

- **_bam** (*str*) – input bam file
- **_bin_config.samtools.binary** (*str*) – samtools binary

Return type str

index(*input_bam=None*) → str

Create a bam index

Variables

- **_bam** (*str*) – input bam file
- **_bin_config.samtools.binary** (*str*) – samtools binary

Returns bai file

Return type str

quick_check() → bool

Checks bam integrity

Variables

- **_bam** (*str*) – input bam file
- **_bin_config.samtools.binary** (*str*) – samtools binary

Returns True if bam is correct, otherwise return False

Return type bool

remove_duplicates(*output_dir=None*)

Remove PCR/Optical duplicates

Variables

- **_bam** (*str*) – input bam file
- **_docker_config.docker** (*str*) – gatk docker image

Returns rmdup bam file, pct_duplicates,

Return type str

Return type

property sample_name: str

Getter sample name from bam RG (Read group). If not found return bam prefix as sample name

Variables

- **_bam** (*str*) – input bam file
- **_bin_config.samtools.binary** (*str*) – samtools binary

Raises

- **ValueError** – when bam header is not found

- **ValueError** – when RG is not available

Return type str

2.3 civic module

class civic.**Civic**(*civic_dir*)

Bases: object

download_latest_release()

property evidences

property genes

get_evidence()

get_variants() → {}

property latest_release

query_cnv(*gene*, *cnvtype*)

query_fusion(*gene1*, *gene2*)

query_variant(*gene*, *variant*, *vartype*, *exon*, *conseq*)

property variants

class civic.**CivicRestAPI**

Bases: object

CIViC class This needs a big refactoring

property accessed_on

This function returns the date in which CIViC was accessed

check_rest_service()

load_civic()

load civic data in memory

query_variant(*gene*, *variant*, *vartype*, *exon*, *conseq*)

return list of evidences given a gene, variant, exon and conseq

civic.**annotate_cna_civic**(*sample_list*, *ann_conf*, *analysis_conf*, *bin_conf*)

Annotate somatic CNAs using CIViC. Create a new annotated vcf and json.

Parameters:

analysis_env [dict] Global dictionary that stores analysis parameters

sample_env [dict] Global dictionary that stores sample data

Returns: Nothing

civic.**annotate_snv_civic**(*sample_list*, *ann_conf*, *dump_json=True*)

Annotate somatic SNV/Indels using CIViC. Create a new annotated vcf and json.

Parameters:

analysis_env [dict] Global dictionary that stores analysis parameters

sample_env [dict] Global dictionary that stores sample data

Returns: sample_list

2.4 cna module

class cna.**CnvKit**(*docker_conf*, *sample_name*, *tumour_bam*, *normals_cnn*, *tumour_purity*, *bed*, *genome*, *threads*, *output_dir*)

Bases: object

batch()

Do all steps.

call_cnv()

property cns_calls_file

create_access_regions()

create_target_regions(*do_target=False*, *do_antitarget=False*)

export_bed()

export_seg()

Export a SEG file (which is compatible with DNAcopy, PureCN and others)

export_vcf()

extract_coverage()

fix()

property ratio_file

segment()

CNV segmentation using CSB as default

2.5 cnv_plot module

class cnv_plot.**CnvPlot**(*cnr_file*, *cns_file*, *calls*, *sample*, *output_dir*, *ref_conf*)

Bases: object

Class for plotting cnvs

plot_cnv(*chr*, *start*, *end*, *gene_list=None*, *add_genes=False*, *offset=None*)

Plot a CNV call, add gene labels (optionally)

plot_genomewide(*genomewide*, *by_chr*)

cnv_plot.**remove_bed_header**(*file*, *pattern*)

2.6 fastq module

class fastq.**Fastq**(*fq*, *expect_paired=True*, *force_naming_convention=True*)

Bases: object

Fastq class. This class automatically validates fastq nomenclature and file consistency

Parameters **fq** (*str*) – fastq file

check_consistency() mean_read_length() check_nomenclature()

check_consistency() → bool

Check that fastq1 and fastq2 are consistent:

- Check that SEQ and QUAL have equal lengths
- Check that fastq1 and fastq2 have equal read number

Parameters

- **fastq1** (*str*) – fastq1 file to be checked
- **fastq2** (*str*) – fastq2 file to be checked

Returns returns True if no issues were detected, otherwise False

Return type bool

check_nomenclature() → bool

Validate a fastq file:

- Check FASTQ valid file extension (fq, fasta, fq.gz, fasta.gz)
- Check that names are equal between fq1 and fq2,
- Check illumina's nomenclature (_S[0-9]+_L[0-9]+_R[12]_[0-9]+)

If paired:

Parameters

- **self.fq1** – fastq1 to be checked
- **self.fq2** – fastq2 to be checked

Type str

Type str

If single-ended:

Parameters **self.fq** – fastq to be checked

Type str

Returns True if fastq(s) follow the Illumina's nomenclature

Return type bool

property fq1: str

Getter Returns fastq1

property fq1_basename: str

Getter Returns fastq1 basename or prefix

property fq2: str

Getter Returns fastq2

property fq2_basename: str

Getter Returns fastq2 basename or prefix

property mean_read_length: int

Calculate the mean read length in base pairs

Parameters **fastq** – fastq file to be checked

Type str

Returns returns the mean read length

Return type int

property sample_name: str

Getter sample name from fastq prefix

Variables **fq** (*str*) – input fastq file

Raises **ValueError** – if fastq name cannot be splitted by "_"

Return type str

exception fastq.FastqNotFound

Bases: Exception

exception fastq.InvalidFastqFile(*fq*)

Bases: Exception

Custom exception to track FASTQ invalid files :param str fq: fastq file

exception fastq.InvalidFastqNomenclature(*fq*)

Bases: Exception

Custom exception to track FASTQ invalid nomenclature :param str fq: fastq file

exception fastq.MissingFastqPair

Bases: Exception

2.7 gene_panel module

class gene_panel.GenePanel(*panel_name*)

Bases: object

disclaimer_db

alias of modules.sqlite.Disclaimer

property disclaimers

get_roi_info(*chr*, *start*, *end*)

is_registered() → bool

Function that checks if the panel is registered on the system Returns True if the panel is registered, otherwise returns False

property is_subpanel

panel_content_db

alias of `modules.sqlite.PanelContent`

panel_db

alias of `modules.sqlite.Panel`

panel_isoforms

alias of `modules.sqlite.PanelIsoforms`

property transcripts

Get gene-transcript relationship for the specified panel using a dict where the primary key refers to a gene id and the secondary key to the selected transcript ID.

property version

class gene_panel.**GenePanelAPI**(*panel_name*, *panel_version=None*)

Bases: `object`

property analysis_genes: dict

property biomarkers: dict

property cna: dict

property disclaimers: dict

is_registered() → bool

property latest_version: str

property qc_criteria: dict

2.8 genomic_plots module

class genomic_plots.**IGV**(*igv_exe*, *sample_name*, *bam*, *bed*, *snapshot_dir*)

Bases: `object`

IGV class to handle variant imaging

Parameters

- **igv_exe** (*str*) – igv executable
- **sample_name** (*str*) – sample name
- **bam** (*str*) – input bam file
- **bed** (*str*) – ROI bed file
- **snapshot_dir** (*str*) – output directory where snapshots will be created

create_batch_script() create_snapshots()

create_batch_script(*img*, *squish=True*, *collapse=False*)

Create a batch script (.bat) needed to create multiple images.

Parameters

- **img** (*str*) – picture format
- **squish** (*bool*) – squish tracks
- **collapse** (*bool*) – collapse alignments to fit vertically

create_snapshots()

Create IGV snapshots using a configured batch script

Variables

- **_igv_exe** (*str*) – picture format
- **_batch_file** (*str*) – picture format

class genomic_plots.**SimpleBamSnap**(*bam*, *fasta*, *docker_conf*, *chr*, *start*, *end*, *output_dir*, *output_name*)

Bases: object

generate_view()

load_bam()

2.9 igv module

2.10 lowpass module

lowpass.**annotate_gc**(*input_bed*, *analysis_conf*, *bin_conf*, *ref_conf*, *ann_conf*)

Add gc content, return a dict with gc content

lowpass.**annotate_mappability**(*input_bed*, *analysis_conf*, *bin_conf*, *ref_conf*, *ann_conf*)

lowpass.**assign_genotype_based_on_cn**(*cn*)

Simple genotype assignment based on copy number

lowpass.**bed_to_vcf**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)

Export *acgm.bed to vcf

lowpass.**calculate_ratios**(*sample_list*, *merged_df*)

Calculate bin ratio

lowpass.**call_cnvs**(*sample_list*, *del_threshold*, *dup_threshold*, *z_score*)

Calling CNVs

lowpass.**classify_cnv**(*sample_list*, *analysis_conf*, *bin_conf*)

lowpass.**create_bins**(*analysis_conf*, *bin_conf*, *ann_conf*, *ref_conf*, *bin_size*)

Create a BED file with genomewide bins

lowpass.**do_lowpass**(*args*)

lowpass.**do_ratio_ref**(*row*, *baseline*, *sample*)

lowpass.**do_ratio_same**(*row*, *median_sample*, *sample*)

lowpass.**extract_coverage**(*sample_list*, *lowpass_bins*, *analysis_conf*, *bin_conf*, *gc_dict*, *map_dict*)
Coverage extraction with megadepth

lowpass.**merge_samples_coverage**(*sample_list*, *analysis_conf*, *lowpass_bins*)
Plotting coverage normalization

lowpass.**normalize**(*df*, *sample*, *median_cov*, *fields*)

lowpass.**plot_normalization**(*df*, *sample*, *output_dir*)
Plotting coverage normalization

lowpass.**ratios_to_json**(*ratio_file*, *sample_name*, *output_folder*)
Convert a ratio file (.bed) to json

lowpass.**remove_bed_header**(*file*, *pattern*)

lowpass.**segment_coverage**(*sample_list*, *n_segments*, *alpha*)
segment with CBS

lowpass.**update_lowpass_db**(*sample_list*, *analysis_conf*)

lowpass.**write_vcf_header_template_for_cnv**(*sample_name*, *bam*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)
Create a vcf template. This is required for cyvcf2

2.11 map module

class map.**BWA**(*sample_name*, *fq1*, *fq2*, *ref*, *outdir*, *docker_config*)
Bases: *map.BaseMapper*

BWA-MEM mapper class

Parameters

- **sample_name** (*str*) – sample name extracted from fastq
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **ref** (*str*) – reference genome in fasta format
- **outdir** (*str*) – output directory
- **bin_config** – configuration object for binaries

align(*num_cpu=4*, *sort=True*)

Align with BWA-MEM

Parameters

- **num_cpu** (*int*) – number of CPU's
- **sort** (*bool*) – sort output BAM by coordinates

class map.**BaseMapper**(*sample_name*, *fq1*, *fq2*, *ref*, *outdir*, *docker_config*)

Bases: object

Parent class to handle read alignment processes. This class checks that input fastq files are valid

Parameters

- **sample_name** (*str*) – sample name extracted from fastq
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **ref** (*str*) – reference genome in fasta format
- **outdir** (*str*) – output directory
- **bin_config** – configuration object for binaries

map.**create_list_file**()

DEPRECATED Generates a GATK-compatible list file from a BED file. TODO documentation

:param args: pipeline command line arguments # :param list sample_list: list of sample objects # :returns: list of sample objects # :rtype: list

map.**create_summary_qc_xlsx**(*sample_list*, *analysis_conf*)

Writes an xlsx summarizing QC metrics. This file outlines the folowing NGS metrics:

- Total number of reads (in Millions)
- Mean coverage
- Call rate (%) at different coverage thresholds (1X, 10X, 20X, 30X, 100X, 200X)
- Lost exons at different coverage thresholds (1X, 10X, 20X, 30X, 100X, 200X)
- Enrichment (%)
- PCR/Optical duplicates (%)
- Mean and std. deviation insert size values (bp)

Parameters

- **sample_list** (*list*) – list of sample objects
- **analysis_conf** (AnalysisConfig) – list of sample objects

Return str a summary file in TSV format

map.**launch_mapping**(*args*, *sample_list*)

Main function for mapping, duplicate removal and qc metric extraction

Parameters

- **args** – pipeline command line arguments
- **sample_list** (*list*) – list of sample objects

Returns list of sample objects

Return type list

map.**read_summary_qc_xlsx**(*sample_list*, *analysis_config*)

Read from summary_qc tsv file to get coverage and alignment metrics

Parameters

- **sample_list** (*list*) – list of sample objects
- **analysis_conf** (AnalysisConfig) – list of sample objects

Return list sample_list list of sample objects

map.**summarize_sample_qc**(*sample*, *qc_folder*, *raw_bam_file*, *rmdup_bam_file*, *roi_bed*, *bin_conf*, *docker_conf*)

From a bam, calculate coverage metrics using Mosdepth and return a summary dict and a bed file :param Sample sample: Sample object :param str qc_folder: list of sample objects :param str bam_file: list of sample objects :param str roi_bed: list of sample objects :param BinaryConfig bin_conf: configuration object for binaries :param DockerConfig docker_conf: configuration object for docker images

Returns summary_qc_dict

Return type dict

2.12 params module

class params.**AnalysisConfig**(*args*)

Bases: *params.BaseConfig*

Analysis configuration class that inherits a basic configuration object

property analysis_date: str

property ann_dir: str

property ann_yaml: str

property bin_yaml: str

Returns from arg param or from defaults

property cgi: bool

property civic: bool

property cnvkit: bool

property command: str

property conservation: str

property create_snapshots: bool

property date_time: str

property db_dir: str

property docker_yaml: str

property freebayes: bool

property gatk: bool

property genome_version: str

property gnomad: bool

property input_dir: str

property lab_data: str

property lancet: bool

property manta: bool

property output_dir: str

property output_name: str

property panel_dirname: str

property panel_full_path: str

property panel_list_full_path: str

property panel_list_name: str

property panel_name: str

property ref_dir: str

property ref_yaml: str

property report_language: str

property rm_dup: bool

property seq_analysis: str

Seq analysis can be: targeted, wgs, lowpass, sarscov2

property thousand_genomes: bool

property threads: int

property user_id: str

validate() → bool

property variant_analysis: str

class params.AnnotationConfig(*data*)

Bases: *params.BaseConfig*

Configuration for annotation resources

validate(*dump_messages=True*)

class params.BaseConfig

Bases: object

Base config class that sets default parameters

class params.**BinaryConfig**(*data*)

Bases: *params.BaseConfig*

Configuration for binary utilities.

static get_bin_path(*self*, *program*)

Get the PATH of a program

get_software_versions(*as_dict=True*) → bool

validate(*dump_messages=True*)

class params.**DockerConfig**(*data*, *args*)

Bases: *params.BaseConfig*

Class that sets docker image

property docker

get_bin_path(*program*)

Get the PATH of a program

validate(*dump_messages=True*)

simple check for docker images installed

class params.**GenomeConfig**(*data*, *args*)

Bases: *params.BaseConfig*

Class that sets genome configuration

property genome_dir

property genome_fasta

validate(*dump_messages=True*)

params.**get_panel_configuration**(*analysis_conf*)

Get panel configuration :param analysis_conf: Analysis configuration object

params.**load_configuration**(*args*, *dump_messages=False*)

Setting analysis parameters: analysis mode, IO directories, number of cpus, analysis date, database directory, annotation resource directory, reference sequence directory, among others. Input parameters are validated here.

params.**set_labdata_env**(*analysis_conf*)

TEMPORARY, we need a petition manager to integrate all data from lab and external Lab data (docx) and sample_data (xlsx) loading. These two files are required to get code relationship (Lab, AP, HC codes) and others like tumour purity estimates,

params.**set_logging**(*args*)

2.13 report module

report.**add_clinical_variant**(*session*, *tables_dict*, *clinical_class*, *sample*, *record*)
Adds a new clinical variant into the main database

report.**annotate_biomarkers**(*sample_list*, *bin_conf*)
Annotate depth information to biomarker positions

report.**igv_snapshots**(*sample_list*, *analysis_conf*, *bin_conf*)
Create a report for somatic variants

report.**launch_report**(*args*, *sample_list*)
Main reporting function

Parameters

- **args** – pipeline command line arguments
- **sample_list** (*list*) – list of sample objects

Returns list of sample objects

Return type list

report.**somatic_report**(*sample_list*, *analysis_conf*, *bin_conf*, *ann_conf*, *docker_conf*)
Create a report for somatic variants

report.**update_cnas**(*sample*, *analysis_conf*, *cna_table*, *session*, *cna_bed*)

report.**update_global_biomarkers**(*sample*, *analysis_conf*)

report.**update_global_cnas**(*sample*, *analysis_conf*, *cna_bed*)

report.**update_global_fusions**(*sample*, *analysis_conf*)
Record all Fusion calls from MANTA

report.**update_global_lost_exons**(*sample*, *analysis_conf*)

report.**update_global_somatic_db**(*sample_list*, *analysis_conf*, *bin_conf*, *ann_conf*, *docker_conf*)

report.**update_global_summary_qc**(*sample*, *analysis_conf*)

2.14 report2 module

report2.**annotate_biomarkers**(*sample_list*, *bin_conf*)
Annotate depth information to biomarker positions

report2.**beautify_info**(*value: str*) → str

Parameters **str** (*info*) – input value to be beautyfied

Return info str beautyfied string

report2.**generate_genomic_snapshots**(*sample_list*, *analysis_conf*, *ref_conf*, *docker_conf*)
Create a report for somatic variants

report2.**germline_report**(*sample_list*, *analysis_conf*)
Create a report for germline variants

report2.**launch_report2**(*args*, *sample_list*)

Main reporting function

Parameters

- **args** – pipeline command line arguments
- **sample_list** (*list*) – list of sample objects

Returns list of sample objects

Return type list

report2.**somatic_report**(*sample_list*, *analysis_conf*)

Create a report for somatic variants (March '22)

report2.**somatic_report2**(*sample_list*, *analysis_conf*)

New version adding tier classifications (Jule '22)

report2.**spliceai_effect**(*csq: dict*) → str

report2.**update_global_biomarkers**(*sample*, *analysis_conf*)

report2.**update_global_cnas**(*sample*, *analysis_conf*, *cna_bed*)

report2.**update_global_fusions**(*sample*, *analysis_conf*)

Record all Fusion calls from MANTA

report2.**update_global_lost_exons**(*sample*, *analysis_conf*)

report2.**update_global_somatic_db**(*sample_list*, *analysis_conf*, *bin_conf*, *ann_conf*, *docker_conf*)

report2.**update_global_summary_qc**(*sample*, *analysis_conf*)

2.15 sample module

class sample.**Sample**(*name*)

Bases: object

add(*key*, *value*)

Add a new attribute dinamically

property name

2.16 sqlite module

class sqlite.**AllCnas**(*user_id*, *lab_id*, *ext1_id*, *ext2_id*, *run_id*, *chromosome*, *start*, *end*, *genes*, *svtype*, *ratio*, *qual*, *cn*)

Bases: sqlalchemy.orm.decl_api.Model

chromosome

cn

end

ext1_id

ext2_id

genes

id

lab_id

qual

ratio

run_id

start

svtype

user_id

class sqlite.**AllFusions**(*user_id*, *lab_id*, *ext1_id*, *ext2_id*, *run_id*, *chromosome*, *start*, *end*, *qual*, *svtype*, *read_pairs*, *split_reads*, *vaf*, *depth*, *fusion_partners*, *fusion_source*, *fusion_diseases*, *flanking_genes*)

Bases: sqlalchemy.orm.decl_api.Model

chromosome

depth

end

ext1_id

ext2_id

flanking_genes

fusion_diseases

fusion_partners

fusion_source

id

lab_id

qual

read_pairs

run_id

split_reads

start

svtype

user_id

vaf

class sqlite.**Biomarker**(*ID*, *gene*, *variant*, *exon*, *chr*, *pos*, *end*, *panel*)

Bases: sqlalchemy.orm.decl_api.Model

chr

end

exon

gene

id

panel

pos

variant

class sqlite.**BiomarkerTable**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

chr

depth

end

exon

ext1_id

ext2_id

gene

id

lab_id

panel

pos

run_id

user_id

vaf

variant

class sqlite.**Cna**(*chromosome*, *start*, *end*, *gene*, *genome_version*, *panel_name*, *panel_version*, *dump_therapeutic*, *min_cn*)

Bases: sqlalchemy.orm.decl_api.Model

chromosome

dump_therapeutic

end

gene

genome_version

id

min_cn

panel_name

panel_version

start

class sqlite.**Disclaimer**(*id*, *panel*, *genes*, *methodology*, *analysis*, *lab_confirmation*, *technical_limitations*, *legal_provisions*, *language*)

Bases: sqlalchemy.orm.decl_api.Model

analysis

genes

id

lab_confirmation

language

legal_provisions

methodology

panel

technical_limitations

class sqlite.**Job**(*User_id*, *Job_id*, *Queue_id*, *Date*, *Analysis*, *Panel*, *Samples*, *Status*)

Bases: sqlalchemy.orm.decl_api.Model

Analysis

Date

Id

Job_id

Panel

Queue_id

Samples

Status

User_id

class sqlite.LostExonsTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

call_rate_100X

call_rate_10X

call_rate_1X

call_rate_200X

call_rate_20X

call_rate_30X

chromosome

end

ensg_id

enst_id

exon_number

ext1_id

ext2_id

gene

id

lab_id

probe_group

run_id

start

user_id

class sqlite.LowpassCnv(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

acmg_classification

acmg_keywords

acmg_score

acmg_version

chromosome

cn

dosage_sensitive_genes

end

ext1_id

ext2_id

fold_change

fold_change_zscore

genotype

id

lab_classification

lab_classification_date

lab_confirmation

lab_confirmation_technique

lab_id

log_fold_change

protein_coding_genes

run_id

start

svlen

svtype

user_id

vcf_json

class sqlite.OtherVariantsTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

allele_frequency

blacklist

classification

clinical_trials

consequence

db_detected_freq

db_detected_number

db_sample_count

depth

enst_id

exon

ext1_id

ext2_id

gene

hgvsc

hgvsg

hgvsp

id

intron

lab_id

max_af

max_af_pop

petition_id

read_support

run_id

therapies

tier_catsalut

tumor_type

user_id

validated_assessor

validated_bioinfo

var_json

variant_type

class sqlite.**Panel**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

Id

call_rate_filter

call_rate_perc

enrichment_perc_filter

genome_version

language

last_modified

lost_exons_filter

lost_exons_perc

panel

panel_bed

read_num_filter

size

total_genes

total_rois

variant_call

version

class sqlite.PanelContent(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

chromosome

end

ense_id

ensg_id

ensp_id

enst_id

exon_end

exon_number

exon_start

feature

gene_name

genome_version

id

mane_select

panel

panel_version

probe_group

refseq_id

start

strand

class sqlite.**PanelIsoforms**(*ID*, *CHROMOSOME*, *START*, *END*, *ENSG_ID*, *ENST_ID*, *GENE_NAME*, *GENOME_VERSION*, *PANEL*, *PANEL_VERSION*)

Bases: sqlalchemy.orm.decl_api.Model

chromosome

end

ensg_id

enst_id

gene_name

genome_version

id

panel

panel_version

start

class sqlite.**Petition**(*Petition_id*, *User_id*, *Date*, *AP_code*, *HC_code*, *CIP_code*, *Tumour_pct*, *Volume*, *Conc_nanodrop*, *Ratio_nanodrop*, *Tape_postevaluation*, *Medical_doctor*, *Billing_unit*, *Medical_indication*, *Date_original_biopsy*)

Bases: sqlalchemy.orm.decl_api.Model

AP_code

Billing_unit

CIP_code

Conc_nanodrop

Date

Date_original_biopsy

HC_code

Id

Medical_doctor

Medical_indication

Petition_id

Ratio_nanodrop

Tape_postevaluation

Tumour_pct

User_id

Volume

class sqlite.PipelineDetailsTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

bwa_version

cgi_version

chimerkb_version

civic_version

cnvkit_version

dbnsfp_version

dbscsnv_version

fastp_version

gatk_version

genome_version

gnomad_version

id

manta_version

pipeline_version

run_id

samtools_version

thousand_genomes_version

vep_version

class sqlite.RareVariantsTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

allele_frequency

blacklist

classification

clinical_trials

consequence

db_detected_freq

db_detected_number

db_sample_count

depth

enst_id

exon

ext1_id

ext2_id

gene

hgvsc

hgvsg

hgvsp

id

intron

lab_id

max_af

max_af_pop

petition_id

read_support

run_id

therapies

tier_catsalut

tumor_type

user_id

validated_assessor

validated_bioinfo

var_json

variant_type

class sqlite.SampleTable(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

analysis_date

bam

cnv_json

date_original_biopsy

diagnosis

ext1_id

ext2_id

ext3_id

extraction_date

id

lab_id

latest_report_pdf

medical_address

medical_center

merged_vcf

panel

petition_id

physician_name

report_db

report_pdf

roi_bed

run_id

sample_db_dir

sample_type

sex

software

software_version

subpanel

tumour_purity

user_id

class sqlite.SampleVariants(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

ann_id

ann_json

ann_key

classification

confirmation_technique

id

lab_confirmation

sample_id

var_id

class sqlite.SummaryQcTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

ext1_id

ext2_id

fastp_json

id

lab_id

petition_id

run_id

summary_json

user_id

class sqlite.TherapeuticTable(**kwargs)

Bases: sqlalchemy.orm.decl_api.Model

allele_frequency

blacklist

classification

clinical_trials

consequence

db_detected_freq

db_detected_number

db_sample_count

depth

enst_id

exon

ext1_id

ext2_id

gene

hgvsc

hgvsg

hgvsp

id

intron

lab_id

max_af

max_af_pop

petition_id

read_support

run_id

therapies

tier_catsalut

tumor_type

user_id

validated_assessor

validated_bioinfo

var_json

variant_type

class sqlite.**VarAnnotation**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

ann_json

ann_key

id

var_id

class sqlite.**Variants**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Model

alt

blacklist

chromosome

count

gene

genome_version

hgvsc

hgvsg

hgvsp

id

isoform

pos

ref

var_type

sqlite.**init**(*analysis_conf*)

sqlite.**init_ngs_database**(*database: str*)

sqlite.**load_cna**(*pname*)

This function requests the available configured CNAs from the CNA database Parameters:

Panel name (str)

Returns: dict with the requested CNAs

sqlite.**load_panel_biomarkers**(*pname*)

This function requests the available configured biomarkers from the biomarker database Parameters:

Panel name (str)

Returns: dict with the requested biomarkers

sqlite.**load_panel_disclaimers**(*pname*, *lang*)

This function requests the available disclaimers Parameters:

Panel name (str) language (str)

Returns: dict with the requested disclaimers

sqlite.**load_panel_transcripts**(*pname*)

This function requests all transcripts from the ROI database Parameters:

Panel name (str)

Returns: dict with the requested transcripts

sqlite.**load_petitions**()

This function requests all petitions from the petition database

Parameters: Petition class

Returns: list of all petitions

sqlite.**update_summary_db**(*sample_list*, *analysis_conf*)

This function adds sample-wise qc metrics into the summary_qc database

Parameters:

analysis_env [dict] Global dictionary that stores analysis parameters

lab_data [dict] Global dictionary that stores lab sample data

sample_env [dict] Global dictionary that stores sample data

Returns: Nothing

2.17 trimming module

trimming.**fastp**(*sample_name*, *output_dir*, *fq1*, *fq2*, *threads*, *fastp_exe*)

Trim raw FASTQ files using fastp

Parameters

- **sample_name** (*str*) – sample name extracted from fastq
- **output_dir** (*str*) – output directory
- **fq1** (*str*) – raw fastq1 (R1)
- **fq2** (*str*) – raw fastq2 (R2)
- **threads** (*str*) – number of cpu cores
- **fastp_exe** (*str*) – fastp binary location

Returns trimmed_fq1, trimmed_fq2

Return type tuple

trimming.**launch_trimming**(*analysis_conf*, *bin_conf*)

Trim raw FASTQ files

Parameters

- **analysis_conf** – analysis configuration object
- **bin_conf** – binary (thid-party software) configuration object

Type *AnalysisConfig*

Type *BinaryConfig*

Returns list of sample objects

Return type list

2.18 utils module

utils.**check_docker_images**(*command*, *image*)
simple check for docker images installed

utils.**compress_vcf**(*input_vcf*, *bgzip_exe*)
Compress a plain text vcf

utils.**convert_vcf_2_json**(*vcf*)

utils.**convert_vcf_to_dict**(*vcf*)

utils.**create_vep_dict**(*info*)

utils.**decompress_vcf**(*input_vcf*, *gunzip_exe*)
Decompress a bgzipped vcf file

utils.**get_bin_path**(*program*)
Get the PATH of a program

utils.**get_fastq_files**(*input_dir*, *avoid_trimmed=False*)
Get fastq files from input directory

utils.**get_input_files**(*input_dir*, *file_type*)
Get all files (fastq, bam, vcf) from an input directory

utils.**index_vcf**(*input_vcf: str*, *tabix_exe: str*) → None
Index a bgzipped vcf

utils.**log2_fold_change**(*num*) → float
Convert a numeric value to log2

utils.**long_aminoacid_2_short**(*long_aa: str*) → str

utils.**long_hgvsp_2_short**(*long_hgvsp: str*) → str

utils.**mean_depth_coordinate**(*bam*, *coordinate*, *samtools_exe*)

utils.**num_to_human**(*num*)
Convert huge numbers to human readable. Return in Million units

utils.**prepare_samples_from_bams**(*input_dir*, *output_dir*) → []
Create sample objects from an input directory with bam files

utils.**prepare_samples_from_vcfs**(*input_dir*, *output_dir*) → []
Create sample objects from an input directory with vcf files

utils.**vcf_2_bed**(*vcf*)

utils.**yaml_to_dict**(*yaml_file*) → str
Take a valid yaml file and return a dict. Sanity check

2.19 var_call module

var_call.**append_manta_small_indels**(*sample_list: List[modules.sample.Sample]*, *bin_conf: modules.params.BinaryConfig*) → List[modules.sample.Sample]

var_call.**cnvkit_log2_json**(*cnr_file*, *cnv_vcf*, *sample*, *outdir*)

var_call.**combine_mutect2_lancet**(*sample_list*, *only_lancet_indels=True*)

Take a list of VCF files and a list of caller names (e.g Mutect2, Lancet) and merges variants according to the following conditions: - SNVs are combined using all callers (Union). Mutect2 info is reported. - Only Indels from Lancet are considered if lancet is used.

var_call.**filter_by_orientation_bias**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)

Filter VCF by orientation bias. This kind of bias is typically observed on low quality degraded FFPE samples.

var_call.**get_mutect2_indels_bed**(*sample_list*, *offset=50*)

Create a BED file with Indel coordinates to be refined.

var_call.**launch_variant_calling**(*args*, *sample_list*)

Main function for deploying variant calling steps :param args: pipeline command line arguments :param list sample_list: list of sample objects :returns: list of sample objects :rtype: list

var_call.**run_cnvkit**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)

Run CNV analysis with CNVkit :param list sample_list: list of sample objects :param AnalysisConfig analysis_conf: analysis configuration object :param BinaryConfig bin_conf: binary configuration object :param GenomeConfig ref_conf: genome configuration object :param DockerConfig docker_conf: docker configuration object :returns: list of sample objects :rtype: list

var_call.**run_decon**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)

Run germline CNV analysis with DECoN :param list sample_list: list of sample objects :param AnalysisConfig analysis_conf: analysis configuration object :param BinaryConfig bin_conf: binary configuration object :param GenomeConfig ref_conf: genome configuration object :param DockerConfig docker_conf: docker configuration object :returns: list of sample objects :rtype: list

var_call.**run_freebayes**(*sample_list*, *bin_conf*, *ref_conf*, *ann_conf*)

var_call.**run_gatk_hc**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *ann_conf*, *docker_conf*)

Call variants with GATK's HaplotypeCaller

var_call.**run_grapes**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*, *docker_conf*)

var_call.**run_lancet**(*sample_list*, *analysis_conf*, *bin_conf*, *docker_conf*, *ref_conf*, *ann_conf*)

Refine indels using Lancet. We observed that complex indels are not well resolved by Mutect2.

var_call.**run_manta**(*sample_list*, *analysis_config*, *bin_config*, *ref_config*, *germline=True*, *somatic=False*)

var_call.**run_mutect2**(*sample_list*, *analysis_config*, *bin_config*, *ref_config*, *ann_config*, *docker_config*)

var_call.**run_octopus**(*sample_list*, *analysis_conf*, *bin_conf*, *ref_conf*)

Call variants with Octopus

2.20 vep_vcf module

class vep_vcf.CSQ

Bases: object

class vep_vcf.Record(*variant_str: str*, *header=None*)

Bases: object

Variant record representation from a vcf

Parameters **variant_str** – VCF entry as string

Type str

property ALT: str

property CHROM: str

property END

property FILTER: str

property FORMAT: []

property GT: list

property ID: str

property INFO: dict

property POS: str

property QUAL: str

property REF: str

property SAMPLE: str

add_info_field(*key: str*, *value: str*, *custom: bool*)

as_dict() → {}

as_string() → str

property civic_items

property clinical_directions

property clinical_significance

property clinical_trials

property clinical_trials_str

property copy_number

property depth

property discordant_reads

property diseases

property diseases_str

property fold_change

property genotype: str

get_clinical_trials()

get_diseases()

get_therapeutic_drugs()

get_tier_classification(*source*)

is_cna() → bool

Return True if the variant is an insertion

is_deletion() → bool

Return True if the variant is a deletion (not SVTYPE)

is_insertion() → bool

Return True if the variant is an insertion

is_mnv() → bool

Return True if the variant is an SNV

property is_rare: bool

is_snv() → bool

Return True if the variant is an SNV

is_sv() → bool

Return True if the variant is an insertion

property len_alt: int

property len_ref: int

read_info() → dict

property read_support: int

set_info_subfield_from_list(*key*, *value_list*)

set_info_subfield_from_str(*key*, *value_str*)

property split_reads

property svend

property svlen

property therapeutic_drugs

property therapeutic_drugs_str

property vaf

property vartype: str

Return variant type (SNV, Deletion, Insertion, MNV, SV, CNA)

class vep_vcf.**VCFreader**(*vcf_in: str*)

Bases: object

add_info_to_header(*info_dict: dict*) → str

fetch(*chr: str*, *pos: Optional[int] = None*, *end: Optional[int] = None*) → *vep_vcf.Record*

Fetch variants from a chr/region

property header: str

static is_gz(*vcf: str*) → bool

class vep_vcf.**VCFwriter**(*vcf_out*, *template=None*, *compress=False*, *sample_name=None*)

Bases: object

VCF writer class

add_info_to_header(*info_dict*)

close()

property header: str

static is_gz(*vcf*) → bool

write(*record*)

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y