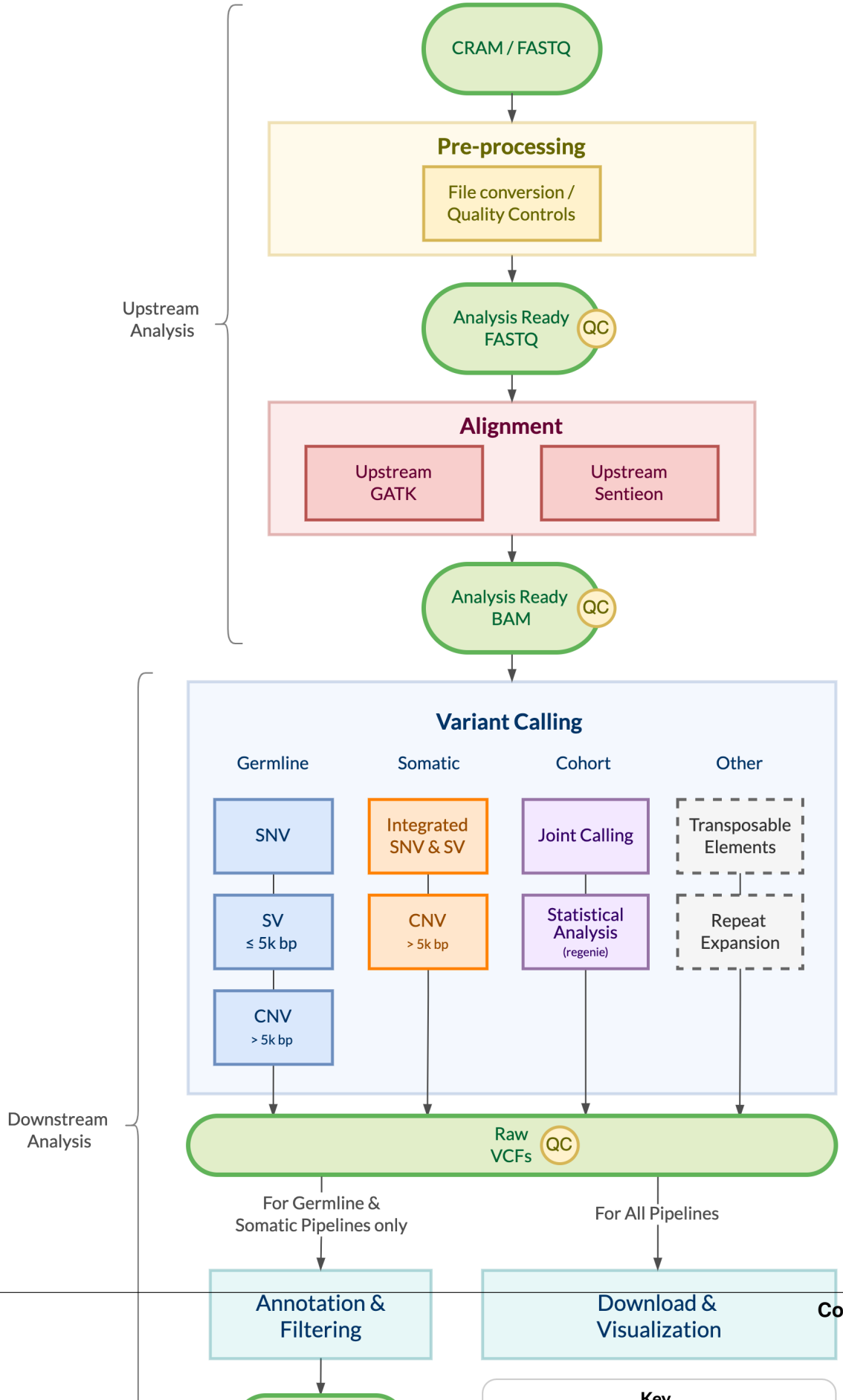
cgap-pipeline-main

Release 2022

Jul 05, 2023

Contents

This is the main documentation for the Computational Genomic Analysis Platform (CGAP) bioinformatics pipelines. The pipelines are developed following a modular approach and different modules can be combined to run a specific set of analyses.


CRAM / FASTQ
Pre-processing
File conversion / Quality Controls
Upstream Analysis
Analysis Ready FASTQ
QC
Alignment
Upstream GATK
Upstream Sentieon
Analysis Ready BAM
QC
Variant Calling
Germline
Somatic
Cohort
Other
SNV
SV ≤ 5k bp
CNV > 5k bp
Integrated SNV & SV
CNV > 5k bp
Joint Calling
Statistical Analysis (regenie)
Transposable Elements
Repeat Expansion
Downstream Analysis
Raw VCFs
QC
For Germline & Somatic Pipelines only
For All Pipelines
Annotation & Filtering
Download & Visualization

The latest stable version for each of the available modules is bundled in the main pipelines repository. Refer to this repository to deploy the pipeline components.

CHAPTER 1

Main Repository

This is the main GitHub repository for the CGAP bioinformatics pipelines (https://github.com/dbmi-bgm/cgap-pipeline-main). The repository bundles the latest stable version for each of the currently available modules.

The repository also contains:

- *MetaWorkflow* objects to describe pipelines that use components from multiple modules
- Basic Docker images that are used as template for most of the module specific images

Finally, there is a README that documents how to install and set up the repository to deploy the pipeline components.

1.1 Docker images

The image `ubuntu-py-generic` is based on Ubuntu 20.04 and contains (but is not limited to) the following software packages:

- python (3.8.12)
- OpenJDK (8.0.312)
- Miniconda3

CHAPTER 2

News and Updates

v1.1.0

- Updated `portal-pipeline-utils` (v2.0.0) to take the deployment code for the new YAML standard
- Updated pipeline submodules:
  - cgap-pipeline-base: v1.1.0
  - cgap-pipeline-upstream-GATK: v1.1.0
  - cgap-pipeline-upstream-sentieon: v1.1.0
  - cgap-pipeline-somatic-sentieon: v1.1.0
  - cgap-pipeline-SNV-germline: v1.1.0
  - cgap-pipeline-SNV-somatic: v1.1.0
  - cgap-pipeline-SV-germline: v1.1.0
  - cgap-pipeline-SV-somatic: v1.1.0
- Conversion of portal objects to the new YAML standard
- Updated and improved documentation

v1.0.0

- Initial release

The modules currently available:

CHAPTER 3

CGAP Pipeline - Base

This is the documentation for the CGAP Pipelines base module (https://github.com/dbmi-bgm/cgap-pipeline-base).

3.1 Overview - Base

The CGAP Pipelines base module (https://github.com/dbmi-bgm/cgap-pipeline-base) contains the CWL description files, Dockerfiles, *Workflow*, *MetaWorkflow* and other shared CGAP Portal objects necessary to run general pipelines (e.g., MD5 Hash and FastQC) and format conversions (e.g., `cram` or `bam` to paired-end `fastq` files). This module is necessary for general CGAP Portal functionality and should always be included when deploying pipelines to a new account.

3.1.1 Docker Images

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `md5` image contains (but is not limited to) the following software packages:

- md5sum (8.25)

The `fastqc` image contains (but is not limited to) the following software packages:

- fastqc (0.11.9)

The `base` image contains (but is not limited to) the following software packages:

- samtools (1.9)
- cramtools (0b5c9ec)
- bcftools (1.11)
- pigz (2.6)
- pbgzip (2b09f97)

The `gatk_picard` image contains (but is not limited to) the following software packages:

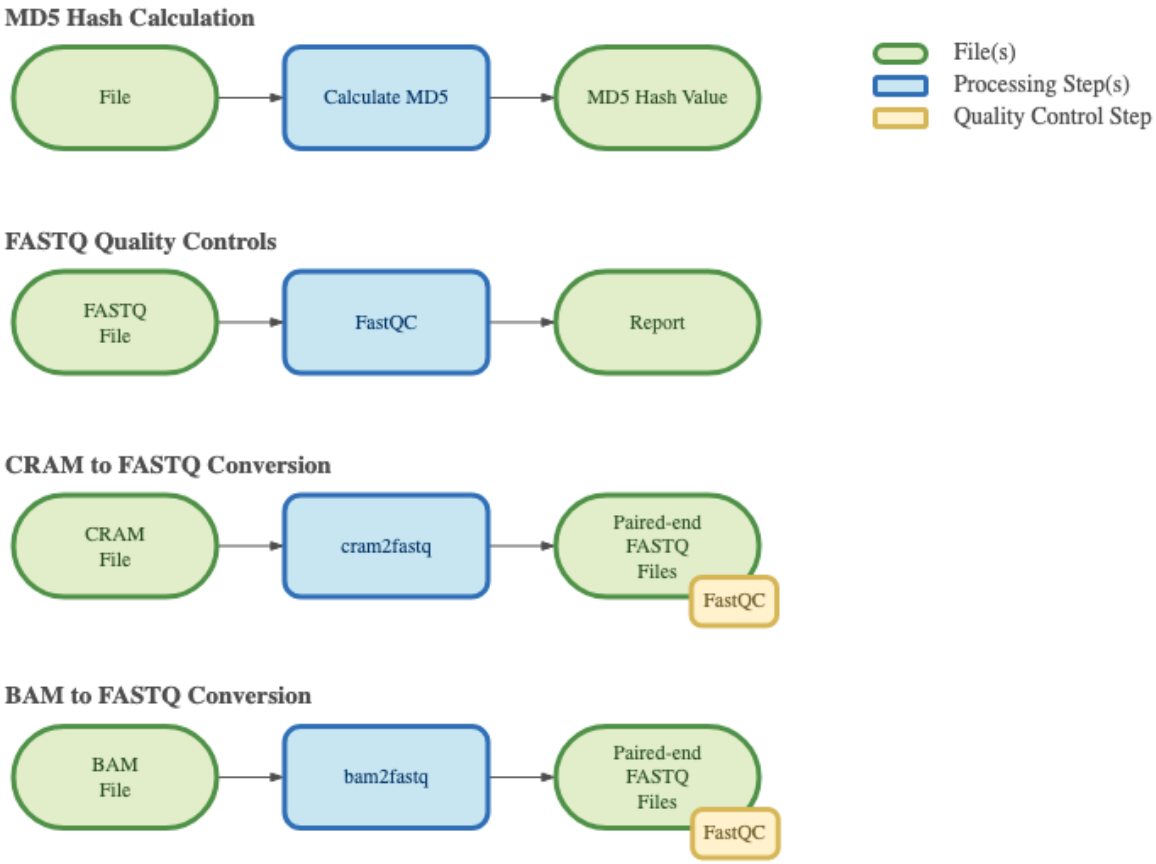
- gatk4 (4.2.6.1)
- picard (2.26.11)

The `granite` image contains (but is not limited to) the following software packages:

- granite-suite (0.2.0)

3.1.2 Pipelines Overview

Pipelines currently available in the module.



3.1.3 Pipelines Description

MD5 Hash

All the files that are uploaded to the CGAP Portal and all the files generated by the pipelines run a MD5 Hash check to ensure file integrity.

- CWL: md5.cwl

FastQC

All `fastq` files generated by format conversion steps (e.g., `cram2fastq`) or uploaded to the CGAP Portal run FastQC to check the sequencing quality and generate a report that is available to the users.

- CWL: fastqc.cwl

FastQC.

CRAM to FASTQ

Pipeline to convert a `cram` file to paired-end `fastq` files. The pipeline runs `cram_to_fastq.sh` and is based on CRAMTools and Samtools software.

- CWL: cram2fastq.cwl

CRAMTools. Samtools.

BAM to FASTQ

Pipeline to convert a `bam` file to paired-end `fastq` files. The pipeline runs `bam_to_fastq.sh` and is based on Samtools software.

- CWL: bam2fastq.cwl

Samtools.

LiftoverVcf

Base implementation of GATK `LiftoverVcf` to convert coordinates for variants provided in `vcf` format to a different genome build.

- CWL: gatk_liftover.cwl

hg19/GRCh37 to hg38/GRCh38

We have a custom implementation for the pipeline with an additional pre-processing step to convert coordinates from **hg19/GRCh37** to **hg38/GRCh38**. The pre-processing uses `preprocess_liftover.py` script to:

- Check if sample identifiers in the `vcf` file match a list of expected identifiers
- Exclude non-standard chromosomes and contigs (e.g., GL000225.1)
- Format chromosome names by adding `chr` prefix if missing (i.e., **hg19** uses only numbers for the main chromosomes). This is necessary as the chain file coordinates use `chr` based names for chromosomes.

The lift-over step uses the `hg19ToHg38.over.chain.gz` chain file (https://cgap-annotations.readthedocs.io/en/latest/liftover_chain_files.html#hg19-grch37-to-hg38-grch38) to map the coordinates between the two builds.

- CWL: workflow_gatk_liftover.cwl

GATK LiftoverVcf.

3.2 News and Updates - Base

3.2.1 Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- Added `sanitize_vcf.py` script to clean `vcf` files
- Added BCFtools `merge` to merge multiple `vcf` files
- Added conversion from `bam` to `fastq`
- Added GATK `LiftoverVcf`
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v27 -> v1.0.0, we are starting a new more comprehensive versioning system

CHAPTER 4

Upstream Pipelines

Upstream pipelines start from raw paired-end sequencing data as `fastq` files, align the reads to the reference genome and produce analysis-ready `bam` files for use in downstream variant calling pipelines. Analysis-ready `bam` files are standard alignment files that undergo additional processing to remove duplicate reads and recalibrate base quality scores.

4.1 CGAP Pipeline - Upstream GATK

This is the documentation for the CGAP Pipelines module for upstream processing with GATK. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-upstream-GATK.

4.1.1 Overview - Upstream GATK

The CGAP Pipelines module for upstream processing with GATK (https://github.com/dbmi-bgm/cgap-pipeline-upstream-GATK) is our open-source solution for processing Whole Genome Sequencing (WGS) and Whole Exome Sequencing (WES) datasets.

The pipeline takes paired-end `fastq` files and produces analysis-ready `bam` files that can be used by any of the CGAP Pipelines downstream modules (e.g., SNV Germline and SV Germline).

The pipeline is based on **hg38/GRCh38** genome build and is optimized for 30x coverage for WGS data and 90x coverage for WES data. It has been tested with WGS data up to 80-90x coverage and WES data ranging from 20 to 200x coverage.

Both the WES and WGS configurations are mostly based on bwa and GATK4, following GATK best practices.

Note: If the user wants to provide `cram` files as input, they must first be converted to paired-end `fastq` files using the CGAP Pipelines base module (https://github.com/dbmi-bgm/cgap-pipeline-base).

Docker Image

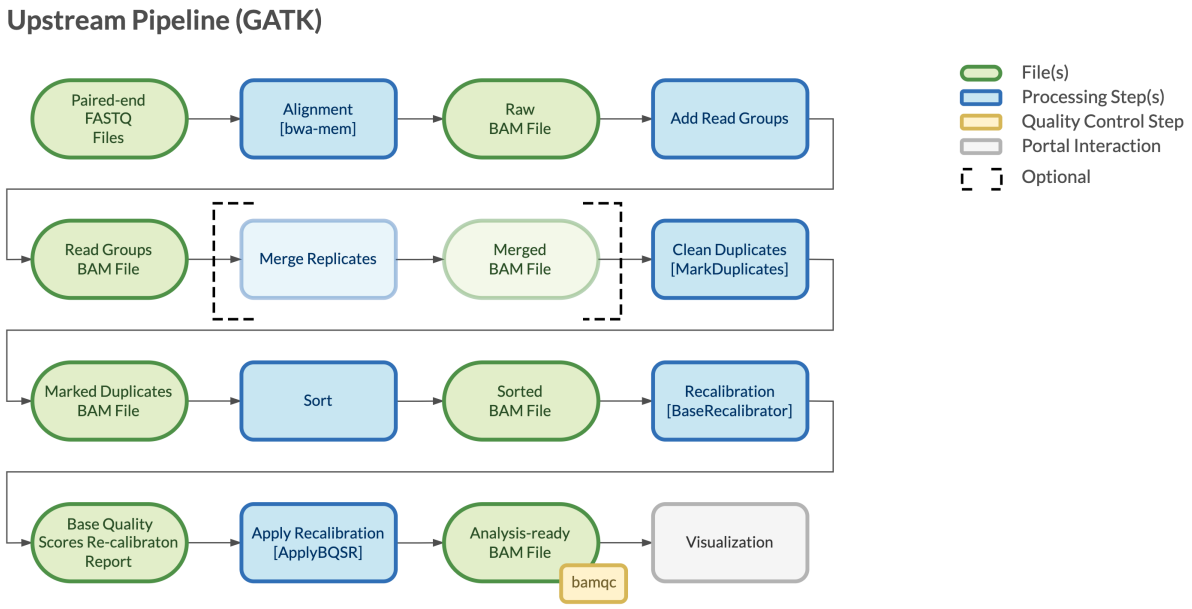
The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `upstream_gatk` image is primarily for **reads alignment and post-processing of the aligned reads**. This image contains (but is not limited to) the following software packages:

- gatk (4.2.6.1)
- picard (2.26.11)
- samtools (1.9)
- bwa (0.7.17)

Pipeline Flow

The overall flow of the pipeline is shown below:



Pipeline Steps

Steps - Upstream GATK

Alignment

This step uses bwa `mem` algorithm to align a set of paired-end `fastq` files to the reference genome. We currently use build **hg38/GRCh38**, more information here. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_bwa-mem-to-bam_no_unzip_plus_integrity-check.cwl

Add Read Groups

This step uses `AddReadGroups.py` (https://github.com/dbmi-bgm/cgap-scripts) to add read groups information to the input `bam` file based on lanes and flow-cells identifiers. The script extracts read groups information from read names and tags the reads accordingly. Unlike Picard `AddOrReplaceReadGroups`, which assumes a single read group throughout the file, the script can handle files that contains a mix of multiple lanes and flow-cells. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_add-readgroups_plus_integrity-check.cwl

Merge BAMs

This step uses Samtools `merge` to merge multiple `bam` files when data comes in replicates. If there are no replicates, this step is skipped. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_merge-bam_plus_integrity-check.cwl

Mark Duplicates

This step uses Picard `MarkDuplicates` to detect and mark PCR duplicates. It creates a duplicate-marked `bam` file and a report with duplicate stats. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_picard-MarkDuplicates_plus_integrity-check.cwl

Sort BAM

This step uses Samtools `sort` to sort the input `bam` file by genomic coordinates. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_sort-bam_plus_integrity-check.cwl

Base Recalibration Report (BQSR)

This step uses GATK `BaseRecalibrator` to create a base quality score recalibration report for the input `bam` file.

- CWL: gatk-BaseRecalibrator.cwl

Apply BQSR and Indexing

This step uses GATK `ApplyBQSR` to apply the base quality score recalibration report to the input `bam` file. This step creates a recalibrated `bam` file and a corresponding index. The output `bam` file is checked for integrity to ensure there is a properly formatted header and the file is not truncated.

- CWL: workflow_gatk-ApplyBQSR_plus_integrity-check.cwl

References

bwa. GATK & Picard Tools. Samtools.

References

bwa. GATK4.

4.1.2 QC - Upstream GATK

This is the documentation for the quality controls that are part of the CGAP Pipelines module for upstream processing with GATK.

BAM Quality Control

Overview

To evaluate the quality of a `bam` file, different metrics are calculated using the custom script `bamqc.py`.

The metrics currently available are:

- Mapping stats
  - Total reads
  - Reads with both mates mapped
  - Reads with one mate mapped
  - Reads with neither mate mapped
- Read length
- Coverage

Definitions

Mapping Statistics

The number of reads (not alignments) are counted as the number of unique read pairs (i.e., if a read pair is mapped to multiple locations it is only counted once).

Coverage

Coverage (=Depth of Coverage) is calculated as below:

```
{ (number of reads w/ both mates mapped) * (read length) * 2 + (number of reads w/␣
↪one mate mapped) * (read length) } / (effective genome size)
```

Here, the effective genome size is the number of non-N bases in the genome for WGS and an estimation of mappable space (exon and UTR regions) for WES.

4.1.3 News and Updates - Upstream GATK

Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v27 -> v1.0.0, we are starting a new more comprehensive versioning system

4.2 CGAP Pipeline - Upstream Sentieon

This is the documentation for the CGAP Pipelines module for upstream processing with Sentieon. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-upstream-sentieon.

4.2.1 Overview - Upstream Sentieon

The CGAP Pipelines module for upstream processing with Sentieon (https://github.com/dbmi-bgm/cgap-pipeline-upstream-sentieon) is our *license-based* option for processing Whole Genome Sequencing (WGS) and Whole Exome Sequencing (WES) datasets.

The pipeline takes paired-end `fastq` files and produces analysis-ready `bam` files that can be used by any of the CGAP Pipelines downstream modules (e.g., SNV Germline and SV Germline).

The pipeline is based on **hg38/GRCh38** genome build and is optimized for 30x coverage for WGS data and 90x coverage for WES data. It has been tested with WGS data up to 80-90x coverage and WES data ranging from 20 to 200x coverage.

The pipeline is based on Sentieon implementation of bwa and GATK4 algorithms, following GATK best practices. Sentieon offers a faster and more computationally efficient implementation of the original algorithms.

Note: If the user wants to provide `cram` files as input, they must first be converted to paired-end `fastq` files using the CGAP Pipelines base module (https://github.com/dbmi-bgm/cgap-pipeline-base).

Docker Image

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

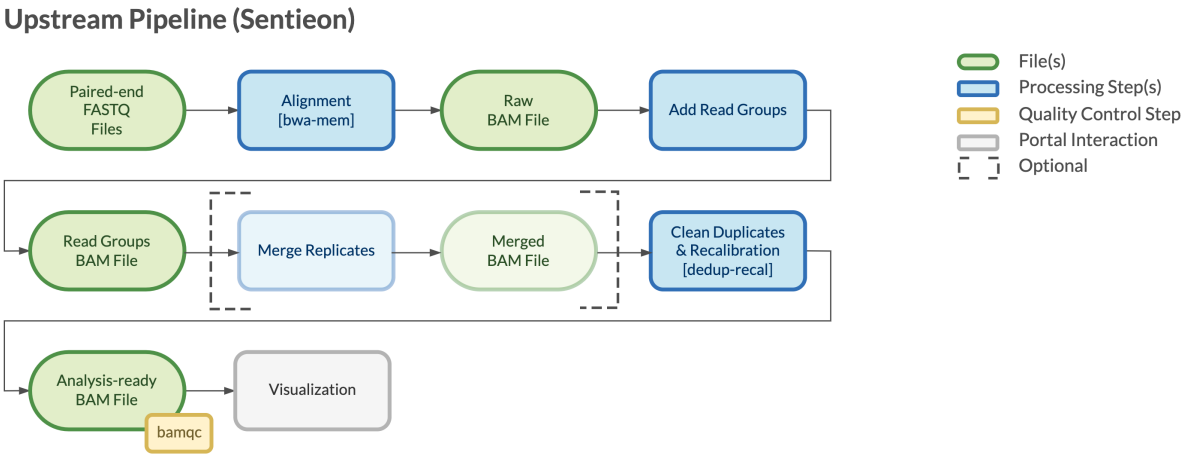
The `upstream_sentieon` image contains (but is not limited to) the following software packages:

- Sentieon (202112.01)
- samtools (1.9)

Pipeline Flow

Our implementation offers an end-to-end solution to process raw sequencing data, producing an analysis-ready `bam` file as described here. For more details see the documentation for Upstream GATK module. This pipeline follows the same logic and structure and simply replaces some of the open-source software with the equivalent Sentieon implementation.

The overall flow of the pipeline is shown below:



References

Sentieon. bwa. GATK4.

4.2.2 News and Updates - Upstream Sentieon

Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v27 -> v1.0.0, we are starting a new more comprehensive versioning system

CHAPTER 5

Downstream Pipelines

Downstream pipelines use analysis-ready `bam` files to generate filtered lists of variants in `vcf` format. These `bam` files are standard alignment files that have been further processed to eliminate duplicate reads and recalibrate base quality scores as described here.

5.1 Germline

Germline pipelines are designed to identify rare inherited variations in an individual's genome. These mutations may be present in the individual's parents or may arise as new mutations in germ cells and be passed on as *de novo* mutations.

5.1.1 CGAP Pipeline - SNV Germline

This is the documentation for the CGAP Pipelines module for Single Nucleotide Variants (SNVs) in germline data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-SNV-germline.

Overview - SNV Germline

The CGAP Pipelines module for germline Single Nucleotide Variants (SNVs) (https://github.com/dbmi-bgm/cgap-pipeline-SNV-germline) processes Whole Genome Sequencing (WGS) and Whole Exome Sequencing (WES) data starting from analysis-ready `bam` files, and produces `g.vcf` and `vcf` files containing SNVs and short Insertions and Deletions (INDELs) as output.

The pipeline supports analysis-ready `bam` files generated by mapping raw reads from both WGS and WES sequencing runs to **hg38/GRCh38** genome build. It can receive the initial `bam` file(s) from either of the CGAP Upstream modules. The pipeline can also receive `vcf` file(s) directly as initial input. **hg38/GRCh38** `vcf` file(s) are supported out-of-the box. **hg19/GRCh37** `vcf` file(s) require an extra step to lift-over the coordinates to **hg38/GRCh38** genome build (https://github.com/dbmi-bgm/cgap-pipeline-base).

The WGS configuration is designed for a trio analysis with proband diagnosed with a likely monogenic disease. It is optimized for data with 30x coverage and has been tested with data up to 80-90x coverage. It can also be run in proband-only, and family modes. The WES configuration is a recent extension of the WGS pipeline, which allows for the processing of WES data. It is optimized for 90x coverage and tested with data ranging from 20 to 200x coverage.

Docker Images

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `snv_germline_gatk` image is primarily for **calling and genotyping variants**. This image contains (but is not limited to) the following software packages:

- gatk (4.2.6.1)

The `snv_germline_granite` image is primarily for **filtering and annotating variants**. This image contains (but is not limited to) the following software packages:

- granite (0.2.0)
- samtools (1.9)

The `snv_germline_misc` image is primarily for **pipeline utilities**. This image does not use the base image provided in the CGAP Pipelines main repository, as some of the software requires an older version of Python. This image contains (but is not limited to) the following software packages:

- python (3.6.8)
- bamsnap-cgap (0.3.0)
- peddy (0.4.7)
- granite (0.2.0)

The `snv_germline_tools` image is primarily for **pipeline utilities**. This image contains (but is not limited to) the following software packages:

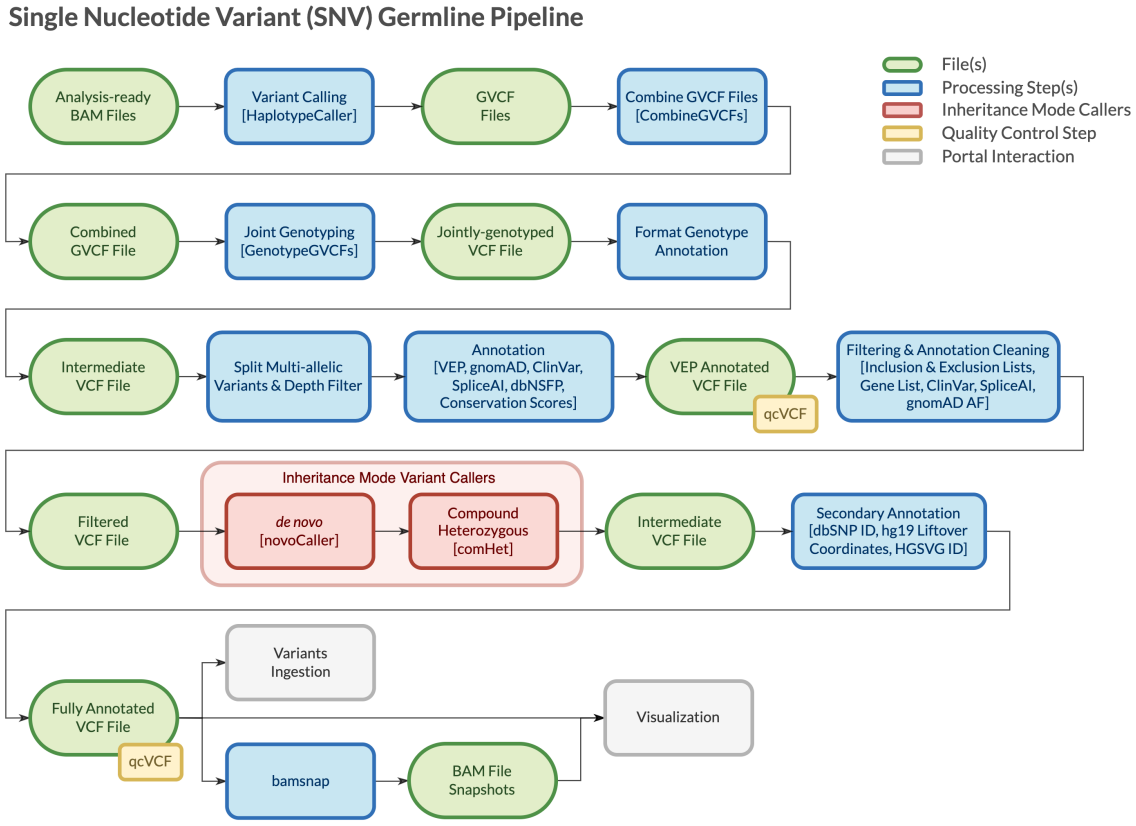
- vcftools (0.1.17, 954e607)
- bcftools (1.11)

The `snv_germline_vep` image is primarily for **annotating variants**. This image contains (but is not limited to) the following software packages:

- vep (101)

Pipeline Flow

The overall flow of the pipeline is shown below:



Pipeline Parts

Both the WGS and WES configurations of the pipeline are mostly based on GATK4, granite, ensembl-vep and bamsnap and are built following GATK best practices. The pipelines:

- Call variants per sample
- Combine the calls to jointly-genotype within a trio or family (if NOT proband-only)
- Annotate and filter the calls
- Refine *de novo* and compound heterozygous calls by running inheritance mode callers
- Generate snapshot images for the final set of variants

`vcf` files are checked for integrity using VCFtools `vcf-validator` at the end of each step during which they are created or modified.

Pipeline Steps

HaplotypeCaller

HaplotypeCaller for WGS

This step uses GATK `HaplotypeCaller` to call SNVs and INDELs via local re-assembly of haplotypes for Whole Genome Sequencing (WGS) data. The software creates a `g.vcf` file from the input `bam` file.

- CWL: gatk-HaplotypeCaller.cwl

HaplotypeCaller for WES

This step uses GATK `HaplotypeCaller` to call SNVs and INDELs via local re-assembly of haplotypes for Whole Exome Sequencing (WES) data. The software creates a `g.vcf` file from the input `bam` file.

To run `HaplotypeCaller` on exomes, we use a custom region file following GATK best practices for WES analysis (reference). We currently use a very permissive region file created for **hg38/GRCh38** to include all exons and UTR regions that are annotated in ensembl (https://cgap-annotations.readthedocs.io/en/latest/exome_regions.html).

- CWL: gatk-HaplotypeCaller_exome.cwl

References

GATK HaplotypeCaller.

CombineGVCFs

This step uses GATK `CombineGVCFs` to merge multiple `g.vcf` files and combines variants.

- CWL: gatk-CombineGVCFs.cwl

References

GATK CombineGVCFs.

GenotypeGVCFs

This step uses GATK `GenotypeGVCFs` to genotype variant calls generated by `HaplotypeCaller`. For a single sample (i.e., proband-only) this creates a `vcf` file from the `g.vcf` input file. For multiple samples (i.e, trio or family), variant calls combined by `CombineGVCFs` are jointly genotyped and a `vcf` file is created from the combined `g.vcf` input file. The resulting `vcf` file is checked for integrity to ensure that the format is correct and the file is not truncated.

- CWL: workflow_gatk-GenotypeGVCFs_plus_vcf-integrity-check.cwl

References

GATK GenotypeGVCFs.

mpileupCounts

This step uses granite `mpileupCounts` to create a `rck` file from a `bam` input file. This is a pre-requisite step for calling *de novo* mutations.

- CWL: granite-mpileupCounts.cwl

Requirements

The command takes a `bam` file and a genome reference `fasta` file as input. To optimize performance, it is also possible to specify a file containing a list of genomic regions to parallelize the analysis.

Output

The output `rck` file contains read pileup counts information for every genomic position, stratified by allele (REFerence vs ALTernate), strand (ForWard vs ReVerse), and type (SNV, INSertion, DELetion). The `rck` file is then compressed and indexed with tabix.

A few lines from an example `rck` file is shown below:

```
#CHR    POS    COVERAGE    REF_FW    REF_RV    ALT_FW    ALT_RV    INS_FW    INS_RV    DEL_FW␣
↪ DEL_REV
13      1      23          0         0         11        12        0         0         0␣
↪ 0
13      2      35          18        15        1         1         0         0         0␣
↪ 0
```

References

granite.

Archive `rck` Files

This step uses granite to create an archive of `rck` files for a trio. This step is a pre-requisite for calling *de novo* mutations.

- CWL: granite-rckTar.cwl

References

granite.

Add `SAMPLEGENO`

This step is for portal compatibility, and can be skipped for non-portal use cases. The tag `SAMPLEGENO` is added by `samplegeno.py` script (https://github.com/dbmi-bgm/cgap-scripts) to the INFO field of the `vcf` file.

- CWL: samplegeno.cwl

```
##INFO=<ID=SAMPLEGENO,Number=.,Type=String,Description="Sample genotype information.␣
↪Subembedded:'samplegeno':Format:'NUMGT|GT|AD|SAMPLEID'">
```

The tag is used to store the original information about genotypes and allelic depth (`AD`) before splitting multi-allelic to bi-allelic variants. It also offers a unique place for accessing the genotype and `AD` information of all the samples.

Variant Annotation

This step splits multi-allelic variants, re-aligns INDELs, removes variants that do not meet a read depth (`DP`) of 3 in at least one sample, and annotates variants for the input `vcf` file. BCFtools is used for split and realignment, `depth_filter.py` (https://github.com/dbmi-bgm/cgap-scripts) is used to filter variants based on depth, and VEP (Variant Effect Predictor) is used for annotation along with several plug-ins and external data sources.

For more details on annotation sources used, see https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#vep

- CWL: workflow_vep-annot_plus_vcf-integrity-check.cwl

References

ensembl-vep. BCFtools.

Variant Filtering

This workflow filters variants in the input `vcf` file based on annotations. The filtering is mostly implemented using granite. The output `vcf` file is checked for integrity to ensure the format is correct and the file is not truncated.

- CWL: workflow_granite-filtering_plus_vcf-integrity-check.cwl

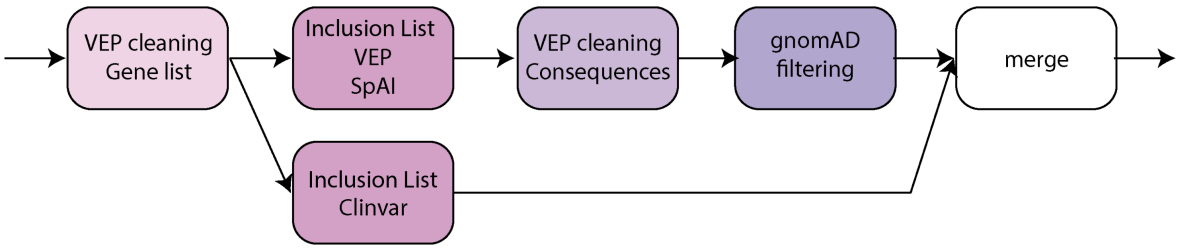
Requirements

The expected input is a single annotated `vcf` file with variant calls. Annotations must include VEP, ClinVar and SpliceAI.

This step can optionally use a panel of unrelated samples in `big` format to filter-out variants with reads supporting an alternate allele in the panel. This option is currently not used in the pipeline. See granite documentation for more information on `big` format.

Steps

The workflow consists of multiple steps as show below.



Gene List

This intermediate step uses granite to clean VEP annotations for transcripts that are not mapping to any gene of interest (the list of genes currently available in the CGAP Portal is here). This step does not remove any variant and only modifies the VEP annotations.

Inclusion List

This intermediate step uses granite to filter-in exonic and functionally relevant variants based on VEP, ClinVar, and SpliceAI annotations. The ClinVar Inclusion list is applied separately and the variants that are rescued do not undergo any further processing by VEP cleaning and Exclusion list.

Criteria to rescue a variant:

- VEP: variant is exonic or annotated as a splice region, and functionally relevant (based on VEP consequences)
- ClinVar: variant annotated as `Pathogenic`, `Likely Pathogenic`, `Conflicting Interpretation of Pathogenicity`, or `Risk Factor`
- SpliceAI: max delta score >= 0.2

VEP Cleaning

This intermediate step uses granite to clean VEP annotations and remove non-relevant consequences. The step eventually discards variants that remain with no VEP annotations after the cleaning.

Exclusion List

This intermediate step uses granite to filter-out common and shared variants based on gnomAD population allele frequency (`AF` > 0.01) and/or a panel of unrelated samples (optional, not used currently).

Merging

This intermediate step merges the set of variants from ClinVar Inclusion list with the other set of fully-filtered variants. For variants that overlap between the two sets, the variant from ClinVar Inclusion list set is maintained to preserve the most complete annotations.

Output

The final output is a filtered `vcf` file containing a subset of variants from the initial `vcf` file. The information attached to filtered variants is the same as in the original variants, with the exception of VEP annotations that have been cleaned to remove non-relevant transcripts and consequences.

References

granite.

de novo Mutations

This step uses granite `novoCaller` to call *de novo* mutations for a trio (**proband**, mother and father). The algorithm handles both SNVs and INDELs and uses allele counts information for the trio and a panel of unrelated individuals to assigning a posterior probability to each variant call. The output `vcf` file is checked for integrity to ensure the format is correct and the file is not truncated. See granite repository and documentation for more information.

- CWL: workflow_granite-novoCaller-rck_plus_vcf-integrity-check.cwl

Requirements

The input is a `vcf` file with genotype information for both the proband and the parents. The software also requires two `rck.tar` files, one for the trio and one for the panel of unrelated individuals. The `rck.tar` files are archives of `rck` files created from the corresponding `bam` files to record allele-specific and strand-specific read counts information.

Output

The step creates an output `vcf` file that contains the same variants as the input file (no line is removed), but with additional information added by the caller (`novoPP` and `RSTR`).

An example:

```
chr1  1041200 .    C    T    573.12 .    AC=2;AF=0.333;AN=6;BaseQRankSum=0.408;DP=76;
↪ExcessHet=3.01;FS=3.873;MLEAC=2;MLEAF=0.333;MQ=60.00;MQRankSum=0.00;QD=13.65;
↪ReadPosRankSum=0.155;SOR=1.877;gnomADgenome=7.00849e-06;SpliceAI=0.11;
↪VEP=ENSG00000188157|ENST00000379370|Transcript|missense_variant|AGRN|protein_coding;
↪novoPP=0.0  GT:AD:DP:GQ:PL:RSTR   0/1:9,4:13:99:100,0,248:6,5,4,2 0/0:34,0:34:96:0,
↪96,1440:23,0,11,0   0/1:12,17:29:99:484,0,309:12,17,2,4   ./.:.:.:.:.:29,0,20,0  ./.
↪:.:.:.:.:19,0,16,0  ./.:.:.:.:.:16,1,22,0  ./.:.:.:.:.:21,0,18,0  ./.:.:.:.:.:28,0,
↪22,0  ./.:.:.:.:.:20,0,24,0  ./.:.:.:.:.:21,0,26,0  ./.:.:.:.:.:11,0,11,0  ./.:.:.:.
↪:.:15,0,13,0  ./.:.:.:.:.:29,0,22,0
```

novoPP

The `novoPP` tag is added to the INFO field of each variants and stores the posterior probability calculated for the variant (0 < `novoPP` <= 1). A high `novoPP` value suggests that the variant is likely to be a *de novo* mutation in the proband.

Notes:

- If the parents have 3 or more alternate reads, `novoCaller` assigns a `novoPP=0` to highlight that the variant is highly unlikely to be a *de novo* mutation
- The model used by `novoCaller` does not fit unbalanced chromosomes, currently we do not report `novoPP` for chrX, Y, or M, except when `novoPP=0`

RSTR

The `RSTR` value is added to each sample genotype and stores the corresponding reads counts by strand at position for reference and alternate alleles used by the caller as `Rf,Af,Rr,Ar` (R: ref, A: alt, f: forward, r: reverse).

References

granite.

Compound Heterozygous Mutations

This step uses granite `comHet` to call compound heterozygous mutations by genes and transcripts, assigning the associate risk based on available annotations. The output `vcf` file is checked for integrity to ensure the format is correct and the file is not truncated.

- CWL: workflow_granite-comHet_plus_vcf-integrity-check.cwl

Requirements

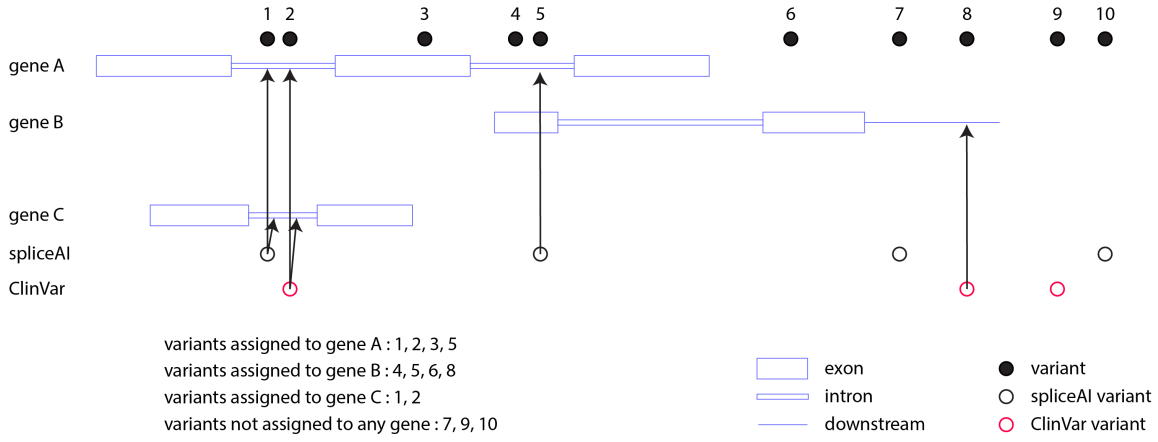
The input `vcf` must have annotations from VEP (consequence, gene and transcript are required). If the annotations also include impact, the software uses it. If not, an existing table calculated for consequences is used instead (https://m.ensembl.org/info/genome/variation/prediction/predicted_data.html).

Specifications

Gene Assignments

To determine compound heterozygous pairs, variants must be first assigned to genes and transcripts. For consistent and inclusive gene assignment, the rules are as follow:

1. Intronic variants are not assigned to the corresponding gene or transcript, with the exception of variants predicted as potential splice sites (SpliceAI) or ClinVar variants
2. Variants that are upstream or downstream of the coding region are not assigned to the corresponding gene or a transcript, with the exception of ClinVar variants



A correct gene assignment is ensured by first 'cleaning' the pre-existing annotations based on spliceAI and ClinVar annotations. This 'cleaning' is performed during the filtering step of the pipeline. This calling step assumes that VEP annotations in the input `vcf` already respect the above rules.

Output

The output `vcf` contains the same variants as the input file (no line is removed), but with additional information added by the caller to variants that are potentially compound heterozygous. This additional information is in the following format:

```
comHet=<Phased_or_Unphased>|<gene_id>|<transcript_id>|<gene_impact_of_comphet_pair>|
↪<transcript_impact_of_comphet_pair>|<mate_variant>
```

An example:

```
comHet=Phased|ENSG00000084636|ENST00000373672~ENST00000488897|STRONG_PAIR|STRONG_
↪PAIR|chr1:31662352G>A
```

A corresponding header is added to the output `vcf` file.

```
##INFO=<ID=comHet,Number=.,Type=String,Description="Putative compound heterozygous␣
↪pairs. Subembedded:'cmpHet':Format:'phase|gene|transcript|impact_gene|impact_
↪transcript|mate_variant'">
```

Gene Versus Transcript

A compound heterozygous pair is defined for each pair of variants and for each gene. If a variant forms a compound heterozygous pair on two or more genes, the output will have a corresponding number of `cmpHet` entries. Compound heterozygous pairs do not always share transcripts, and shared transcript for each pair are listed as additional information in the `transcript` field.

Phase

A compound heterozygous pair is either classified as Phased or Unphased.

Impact

The predicted impact of a compound heterozygous pair is calculated as follows:

1. If VEP impact for both variants is HIGH (H) or MODERATE (M), SpliceAI score >= 0.8 (S) or ClinVar Pathogenic or Likely Pathogenic (C), the pair is called a `STRONG_PAIR`
2. If only one of the variants is H, M, S or C, the pair is called a `MEDIUM_PAIR`
3. If none of the above, the pair is called a `WEAK_PAIR`

The impact is calculated both at the gene level (gene impact) and at the transcript level (transcript impact).

Report

This step also generates a report that provides additional information on the compound heterozygous pairs that are called. The report contains statistics on the total number of pairs and their distribution by genes, transcripts, and predicted impact.

By Genes

For each gene, the program reports the number of compound heterozygous pairs called for the gene (`name`), together with the number of transcripts and variants involved. In each category, it is reported the total number of elements that are involved in a compound heterozygous pair, as well as the total number of elements involved in a pair that is also Phased.

```
"by_genes": [
  {
    "name": "ENSG00000004455",
    "pairs": {
        "phased": 0,
        "total": 1
    },
    "transcripts": {
        "phased": 0,
        "total": 11
    },
    "variants": {
        "phased": 0,
        "total": 2
    }
  },
  ...
]
```

By Transcripts

For each transcript, the program reports the number of compound heterozygous pairs called for the transcript (`name`), together with the number of variants involved and the gene to which the transcript belongs. In each category, it is reported the total number of elements that are involved in a compound heterozygous pair, as well as the total number of elements involved in a pair that is also Phased.

```
"by_transcripts": [
  {
    "name": "ENST00000218200",
    "gene": "ENSG00000102081",
    "pairs": {
       "phased": 3,
```

(continues on next page)

(continued from previous page)

```
        "total": 6
    },
    "variants": {
        "phased": 4,
        "total": 4
    }
  },
  ...
]
```

By Impact

For each impact, the program reports the number of compound heterozygous pairs predicted with that impact (`name`) as the worst possible impact, together with the number of genes, transcripts and variants involved. In each category, it is reported the total number of elements that are involved in a compound heterozygous pair, as well as the total number of elements involved in a pair that is also Phased.

```
"by_impact": [
    {
      "name": "MEDIUM_PAIR",
      "pairs": {
          "phased": 28,
          "total": 44
      },
      "genes": {
          "phased": 23,
          "total": 34
      },
      "transcripts": {
          "phased": 55,
          "total": 81
      },
      "variants": {
          "phased": 51,
          "total": 78
      }
    },
    ...
  ]
```

References

granite.

dbSNP rsID Update

This step uses `parallel_dbSNP_ID_fixer.sh` to run `dbSNP_ID_fixer.py` script to update dbSNP rsIDs in a sample `vcf` file ID column. The output `vcf` file is checked for integrity.

- CWL: workflow_parallel_dbSNP_ID_fixer_plus_vcf-integrity-check.cwl

Requirements

Must be run on input `vcf` following BCFtools `norm` since it only allows one variant per line in the input `vcf`.

Output

This process follows these rules:

1. Variants in the input `vcf` are matched to the reference dbSNP `vcf` by CHROM, POS, REF, and ALT columns
2. All rsIDs in the input `vcf` ID column are discarded and replaced by the IDs found in the reference dbSNP `vcf`
3. Given a known bug where BCFtools `norm` leaves an erroneous rsID at multi-allelic sites, this will sometimes result in replacing an existing (but wrong) rsID with `.`
4. When multiple dbSNP rsIDs exist for a single CHROM, POS, REF, and ALT in the dbSNP reference `vcf`, we include them all separated by `;`. In our testing, we came across multiple cases where the same variant is associated with multiple rsIDs. In some cases, one of these rsIDs is listed as a parent in the dbSNP database (https://www.ncbi.nlm.nih.gov/snp/), but in other cases, it was not possible to find a link between the different rsIDs for variants that appear to be identical. gnomAD (https://gnomad.broadinstitute.org/help) described similar issues with dbSNP within their database. Taking the safer approach, we decided to include all possible rsIDs for a given variant at this stage
5. If a variant has a non-rsID within the ID column, it is not discarded and will be appended at the beginning of the `;`-delimited list of any and all rsIDs from the reference dbSNP `vcf`

An example of how these rules are followed with various inputs is found below:

Sample VCF ID (input)	dbSNP reference VCF ID	Sample VCF ID (output)
.	no variant	.
.	1 variant	rsID
.	3 variants	rsID;rsID;rsID
rsID	no variant	.
rsID	2 variants	rsID;rsID
rsID;rsID	no variant	.
rsID;rsID	4 variants	rsID;rsID;rsID;rsID
otherID	no variant	otherID
otherID	1 variant	otherID;rsID
rsID;otherID;otherID	1 variant	otherID;otherID;rsID

For more details, see https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#dbsnp

hg19/GRCh37 lift-over and HGVSg

This step uses `liftover_hg19.py` (https://github.com/dbmi-bgm/cgap-scripts) and `hgvsg_creator.py` to add **hg19/GRCh37** coordinates and HGVSg entries to qualifying variants from a filtered input `vcf` file. The output `vcf` file is checked for integrity.

- CWL: workflow_hg19lo_hgvsg_plus_vcf-integrity-check.cwl

Requirements

Must be run on input `vcf` following BCFtools `norm` since it only allows one variant per line in the input `vcf` file.

Output

This step creates an output `vcf` file that has the same entries from the input `vcf` file (no line is removed), but with additional information. Five definitions are added to the header:

```
##INFO=<ID=hgvsg,Number=.,Type=String,Description="hgvsg created from variant␣
↪following best practices - http://varnomen.hgvs.org/recommendations/DNA/">
##INFO=<ID=hg19_chr,Number=.,Type=String,Description="CHROM in hg19 using LiftOver␣
↪from pyliftover">
##INFO=<ID=hg19_pos,Number=.,Type=Integer,Description="POS in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
##INFO=<ID=hg19_end,Number=1,Type=Integer,Description="END in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
##INFO=<ID=hgvsg_hg19,Number=1,Type=String,Description="hgvsg for liftover␣
↪coordinates in hg19 created from variant following best practices - http://varnomen.
↪hgvs.org/recommendations/DNA/">
```

The data associated with these tags are also added to the INFO field of the `vcf` for qualifying variants using the following criteria.

For **hg19/GRCh37** lift-over:

1. For the **hg19/GRCh37** lift-over, all variants with successful conversions will include data for both the `hg19_chr=` and `hg19_pos=` tags in the INFO field. Failed conversions (e.g., coordinates that do not have a corresponding region in **hg19/GRCh37**) will not print the tags or any lift-over data
2. Given that pyliftover does not convert ranges, the single-point coordinate in **hg38/GRCh38** corresponding to each variant's CHROM and POS are used as query, and the **hg19/GRCh37** coordinate (result) will also be a single-point coordinate

For HGVSg:

1. For HGVSg, best practices (http://varnomen.hgvs.org/recommendations/DNA/) are followed. All variants on the 23 nuclear chromosomes receive a `g.` and all mitochondrial variants receive an `m.`
2. All variants should receive an `hgvsg=` tag within their INFO field with data pertaining to their chromosomal location and variant type
3. If a variant on a contig (e.g., chr21_GL383580v2_alt) were to be included in the filtered `vcf`, it would not receive an `hgvsg=` tag, or any HGVSg data, since contigs were not included in the python script's library of chromosomal conversions (e.g., chr1 is NC_000001.11)
4. Any variant that receives a value for both `hg19_pos` and `hg19_chr` will also receive an entry for the `hgvsg_hg19=` tag based on this lift-over position. These are calculated identically to the **hg38/GRCh38** HGVSg fields, but with the appropriate **hg19/GRCh37** chromosomal accessions

Note: Although `hg19_end` is written into the header of all `vcf` files, this tag should only appear in the INFO field of structural and copy number variants given the requirement for an `END` coordinate in the INFO block (which is not present in SNVs).

For more details, see https://cgap-annotations.readthedocs.io/en/latest/liftover_chain_files.html#hg38-grch38-to-hg19-grch37 and https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#hgvsg.

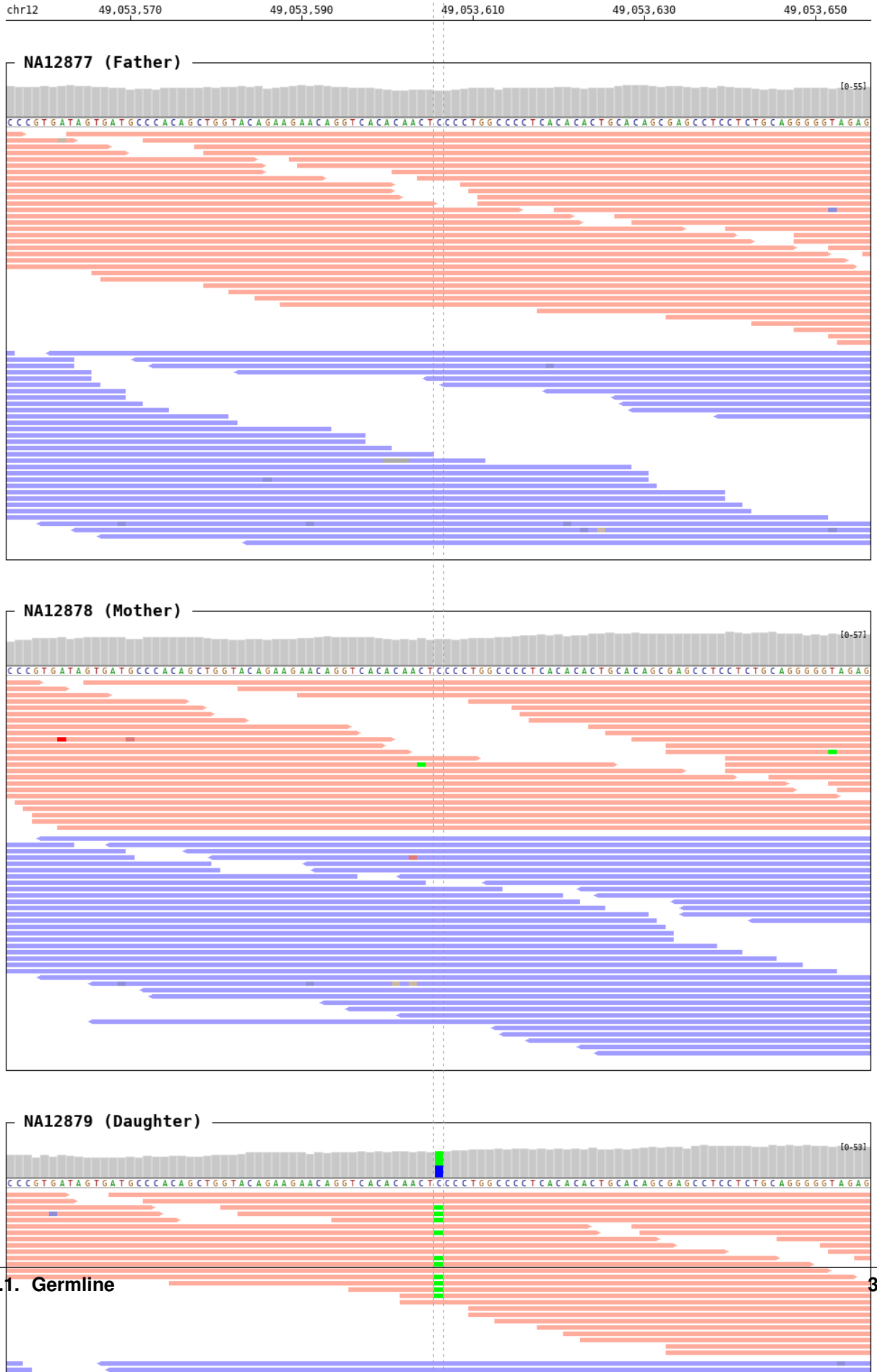
Bamsnap

This step uses bamsnap to generate a zip archive of IGV-like snapshots with reads information for all the variants and the samples in the target `vcf` file.

- CWL: bamsnap.cwl

Output

An example output for a snapshot (`png`):


chr12
49,053,570
49,053,590
49,053,610
49,053,630
49,053,650
NA12877 (Father)
[0-55]
NA12878 (Mother)
[0-57]
NA12879 (Daughter)
[0-53]

References

bamsnap.

References

GATK4. ensembl-vep. bamsnap. granite. VCFtools.

QC - SNV Germline

This is the documentation for quality controls that are part of CGAP Pipelines module for germline Single Nucleotide Variants (SNVs).

VCF Quality Control

Overview

To evaluate the quality of a `vcf` file, different metrics are calculated using granite `qcVCF`. The software calculates both sample-based, as well as, family-based statistics.

The metrics currently available for sample are:

- Variant types distribution
- Base substitutions
- Transition-transversion ratio
- Heterozygosity ratio
- Depth of coverage (GATK)
- Depth of coverage (raw)

The metrics currently available for family are:

- Mendelian errors in trio

For each sample, ancestry and sex are also predicted using peddy[1]. The predicted values allow to identify errors in sample labeling, contaminations events, and other errors that can occur during handling and processing of the sample.

Definitions

Variant Types Distribution

Total number of variants classified by type as:

- **DEL**etion (*ACTG>A or ACTG>**)
- **INS**ertion (*A>ACTG or *>ACTG*)
- **S**ingle-**N**ucleotide **V**ariant (*A>T*)
- **M**ulti-**A**llelic **V**ariant (*A>T,C*)

[1] Pedersen and Quinlan, Who's Who? Detecting and Resolving Sample Anomalies in Human DNA Sequencing Studies with Peddy, The American Journal of Human Genetics (2017), http://dx.doi.org/10.1016/j.ajhg.2017.01.017

- **M**ulti-**N**ucleotide **V**ariant (*AA>TT*)

Base Substitutions

Total number of SNVs classified by the type of substitution (e.g., C>T).

Transition-Transversion Ratio

Ratio of transitions to transversions in SNVs. It is expected to be [2, 2.20] for WGS and [2.6, 3.3] for WES.

Heterozygosity Ratio

Ratio of heterozygous to alternate homozygous variants. It is expected to be [1.5, 2.5] for WGS analysis. Heterozygous and alternate homozygous sites are counted by variant type.

Depth of Coverage

Average depth of all variant sites called in the sample.

Depth of coverage (GATK) is calculated based on `DP` values as assigned by GATK. Depth of coverage (raw) is calculated based on raw read counts calculated directly from the bam file.

Mendelian Errors in Trio

Variant sites in proband that are not consistent with mendelian inheritance rules based on parent genotypes. Mendelian errors are counted by variant type and classified based on genotype combinations in trio as:

Proband	Father	Mother	Type
0/1	0/0	0/0	*de novo*
0/1	1/1	1/1	Error
1/1	0/0	[any]	Error
1/1	[any]	0/0	Error
1/1 | 0/1	./.	[any]	Missing in parent
1/1 | 0/1	[any]	./.	Missing in parent

Ancestry and Sex Prediction

Ancestry prediction is based on projection onto the thousand genomes principal components. Sex is predicted by using the genotypes observed outside the pseudo-autosomal region on X chromosome.

References

granite. peddy.

News and Updates - SNV Germline

Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- Updated `samplegeno.py` script to better support multiple `vcf` files
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v27 -> v1.0.0, we are starting a new more comprehensive versioning system

5.1.2 CGAP Pipeline - SV Germline

This is the documentation for the CGAP Pipelines module for Structural Variants (SVs) in germline data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-SV-germline.

Overview - SV Germline

The CGAP Pipelines module for germline Structural Variants (SVs) (https://github.com/dbmi-bgm/cgap-pipeline-SV-germline) identifies, annotates, and filters SVs starting from analysis-ready `bam` files to produce final sets of calls in `vcf` format.

SVs are a class of large genomic variants that includes deletions, duplications, translocations, inversions and other complex events, generally with a size of 50 bp or longer. SVs are identified by algorithms that seek out aberrantly mapping reads, including read pairs with unexpected fragment sizes, mapping orientations, and hard or soft clipping events (e.g., split reads). SVs are related to another class of large genomic variants, Copy Number Variants (CNVs). CNVs include deletions (also referred to as losses) and duplications (also referred to as gains), that result in a copy number change. CNVs are included in the broader definition of SVs. However, it's useful to maintain a separate classification to account for the differences between the algorithms used for their detection. CNVs are identified by algorithms that search for unexpected variation in sequencing coverage. As a result, algorithms for CNV detection perform better and are more robust in the identification of larger events, as they don't rely on local context information, but are less powerful and accurate than SV algorithms in the detection of smaller events. Given these substantial differences, CGAP implements both SV and a CNV calling algorithms, with the goal to combine the strength of both algorithmic approaches for an integrated analysis of the structural variation in the germline genome.

The pipeline is mostly based on the SV calling algorithm Manta, alongside software for variants annotation and filtering (ensembl-vep, Sansa and granite).

The pipeline is designed for proband-only or trio analysis, with the proband diagnosed with a likely monogenic disease. It can receive the initial `bam` file(s) from either of the CGAP Upstream modules. The pipeline can also receive `vcf` file(s) directly as initial input. **hg38/GRCh38** `vcf` file(s) are supported out-of-the box. **hg19/GRCh37** `vcf` file(s) require an extra step to lift-over the coordinates to **hg38/GRCh38** genome build (https://github.com/dbmi-bgm/cgap-pipeline-base).

For proband-only analysis, a single `bam` file is provided to Manta that runs a *Single Diploid Sample Analysis*, resulting in a `vcf` file containing SVs with genotypes for the proband. For trio analysis, three `bam` files are provided to Manta that runs a *Joint Diploid Sample Analysis*, resulting in a single `vcf` file containing SVs with genotypes for all three individuals.

Note: The pipeline is not optimized for Whole Exome Sequencing (WES) data. If the user is directly providing `bam` file(s) as input, the `bam` file(s) must be aligned to **hg38/GRCh38** for compatibility with the annotation steps. Manta

requires standard paired-end sequencing data and can't run on mate-pair data or data produced with more complex library preparation protocols.

Docker Images

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `manta` image is primarily for **SVs identification**. This image does not use the base image provided in the CGAP Pipelines main repository, as some of the software requires an older version of Python. This image contains (but is not limited to) the following software packages:

- python (2.7.13)
- manta (1.6.0)
- samtools (1.9)

The `sv_germline_granite` image is for **SVs annotation and filtering**. This image contains (but is not limited to) the following software packages:

- granite (0.2.0)
- pyliftover (0.4)

The `sv_germline_tools` image is for **SVs annotation**. This image contains (but is not limited to) the following software packages:

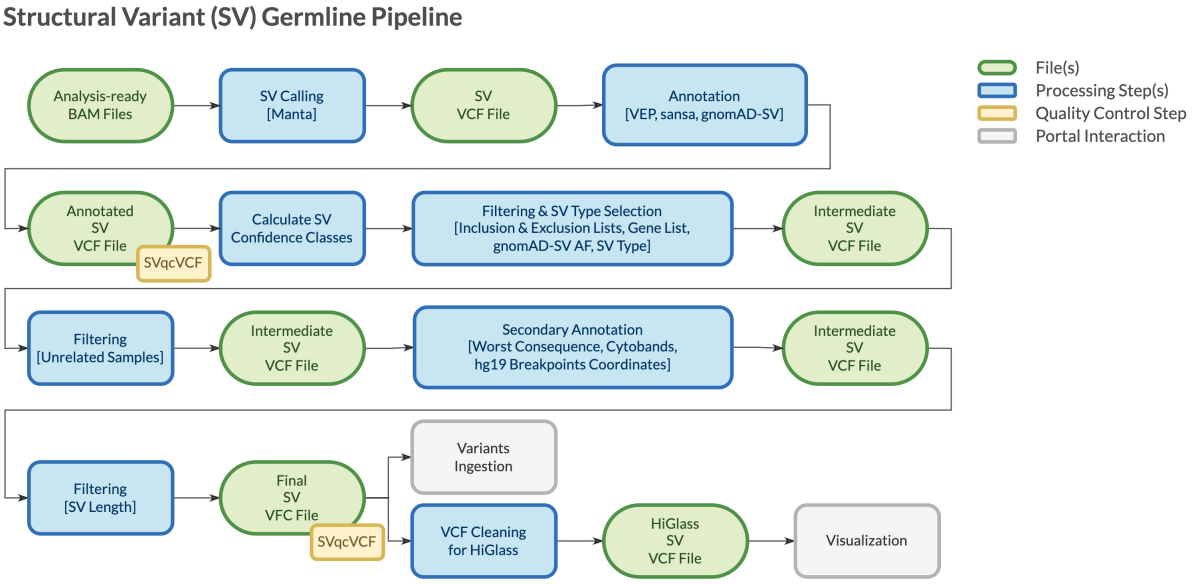
- vcftools (0.1.17, 954e607)
- bcftools (1.11)
- sansa (0.0.8, a30e1a7)

The `sv_germline_vep` image is for **SVs annotation**. This image contains (but is not limited to) the following software packages:

- vep (101)

Pipeline Flow

The overall flow of the pipeline is shown below:



Pipeline Parts

Largely, the pipeline consists of three parts:

Part 1. Starting from analysis-ready `bam` file(s), Manta calls SVs in `vcf` format

Part 2. SVs are annotated using VEP (Ensembl Variant Effect Predictor) to add transcripts information and Sansa to add gnomAD SV allele frequencies

Part 3. SVs are filtered to remove non-functional variants, artifacts, and variants that are common in the population

`vcf` files are checked for integrity using VCFtools `vcf-validator` at the end of each step during which they are created or modified.

Pipeline Steps

Part 1. Manta Structural Variants Identification

Manta

This step executes the `manta.sh` script, which runs the Manta algorithm to identify potential structural variations (SVs) in the input data. The output is a `vcf` file, which is then checked for integrity to confirm that the format is correct and the file is complete.

- CWL: workflow_manta_integrity-check.cwl

Input

The script takes analysis-ready `bam` file(s), generated from either of the CGAP Upstream modules. The input file(s) can also be provided directly by the user.

Running Manta

Manta executes a *Joint Diploid Sample Analysis* when more than one sample is provided, and a *Single Diploid Sample Analysis* when run for a single sample (i.e., proband-only). The algorithm identifies deletions (`SVTYPE=DEL`), duplications (`SVTYPE=DUP`), insertions (`SVTYPE=INS`), and inversion/translocations (`SVTYPE=BND`).

A `callRegions` region file containing a list of chromosome names limits the search to canonical and sex chromosomes (i.e., chr1-chr22, chrX and chrY).

The script `convertInversion.py` (https://github.com/Illumina/manta) is also run to separate (`SVTYPE=INV`) from other translocations (`SVTYPE=BND`).

References

Manta.

Part 2. Structural Variants Annotation

Sansa and VEP

This workflow uses Sansa and VEP (Ensembl Variant Effect Predictor) to annotate the identified structural variants (SVs) and combines the annotations into a single `vcf` file. The resulting `vcf` file is checked for integrity.

- CWL: workflow_sansa_vep_combined_annotation_plus_vcf-integrity-check.cwl

Sansa

`sansa.sh` script executes Sansa to identify SVs that match the **hg38/GRCh38** lift-over of the gnomAD v2 structural variants database (https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#gnomad-structural-variants). The script sorts the input `vcf` file and runs the following command for Sansa:

```
sansa annotate -m -n -b 50 -r 0.8 -s all -d $gnomAD $vcf
```

- `-m` returns all SVs (including those without matches to the database)
- `-n` allows for matches between different SV types (to allow SV to match DEL and DUP)
- `-b 50` is the default parameter for maximum breakpoint offset (in bp) between the newly-identified SV and the SV in gnomAD SV
- `-r 0.8` is nearly the default parameter (default is 0.800000012) for minimum reciprocal overlap between SVs
- `-s all` provides all matches instead of automatically selecting the single best match
- `-d $gnomAD` is the gnomAD SV database
- `$vcf` is the input `vcf` file

VEP

`vep-annot_SV.sh` script executes VEP to annotate the SVs with genes and transcripts. The script produces an annotated `vcf` containing all variants.

A maximum SV size slightly larger than *chr1* in the **hg38/GRCh38** genome (`--max_sv_size 250000000`) is used in the VEP command to avoid filtering large SVs. The `--overlaps` option is also included to record the overlap

between the VEP features and the SVs (reported in bp and percentage). The `--canonical` option is included to flag canonical transcripts.

Combine Sansa and VEP

The outputs from Sansa and VEP are combined using `combine_sansa_and_VEP_vcf.py` script. The `vcf` file generated by VEP is used as a scaffold, onto which gnomAD SV annotations from Sansa are added.

When multiple matches are identified in the gnomAD SV database, the following logic applies to select the best (and rarest) match:

1. Select a type-matched SV (if possible), and the rarest type-matched variant from gnomAD SV (using `AF`) if there are multiple matches
2. If none of the options are a type-match, select the rarest variant from gnomAD SV (using `AF`)

Note: CNV is a variant class in gnomAD SV, but not in the Manta output. Since DELs and DUPs are types of CNVs, we prioritize as follows: (I) we first search for type-matches between DEL and DEL or DUP and DUP, (II) if a type-match is not found for the variant, we then search for type-matches between DEL and CNV or DUP and CNV, (III) all other combinations (e.g., INV and CNV, or DEL and DUP) are considered to **not** be type-matched.

These rules were set given limitations on the number of values the gnomAD SV fields can have for filtering in the CGAP Portal and to avoid loss of rare variants in the upcoming filtering steps. The final output is a `vcf` file with annotations for both gene/transcript and gnomAD SV population frequencies.

Confidence Classes

This workflow assigns a confidence class to each of the SVs identified by the pipeline.

- CWL: manta_add_confidence.cwl

Confidence classes are calculated and assigned using the `SV_confidence.py` script. A single `vcf` is required as input, and the file must contain the information supporting each of the calls created by Manta. The possible confidence classes are:

- HIGH
- MEDIUM
- LOW
- NA (Not Available): assigned only to insertions, which are currently not ingested into the portal

Confidence classes are calculated based on the following parameters:

- *length*: the length of the variant, calculated as the absolute value of the `SVLEN` field
- *split-reads*: the number of alternative split reads, based on the `SR` field
- *spanning-reads*: the number of alternative spanning reads, based on the `PR` field
- *split-read-ratio*: the proportion of alternative split reads out of the total number of reference and alternative split reads, based on the `SR` field
- *spanning-read-ratio*: the proportion of alternative spanning reads out of the total number of reference and alternative spanning reads, based on the `PR` field

For each variant, all the samples are classified according to the following criteria:

High Confidence Calls

```
length > 250bp & split-reads >= 5 & split-read-ratio >= 0.3 & spanning-reads >= 5 &␣
↪spanning-read-ratio >= 0.3
or
length =< 250bp & split-reads > 5 & split-read-ratio > 0.3
```

Note: In the case of translocations, the length parameter is not taken into consideration. These SVs are examined based on the number of split reads and spanning reads and have the same priority as variants which are greater than 250 bp.

Medium Confidence Calls

```
length > 250bp & split-reads >= 3 & split-read-ratio >= 0.3 & spanning-reads >= 3 &␣
↪spanning-read-ratio >= 0.3
or
length =< 250bp & split-reads > 3 & split-read-ratio > 0.3
```

Low Confidence Calls

All the other variants.

The calculated confidence classes are added as the new FORMAT field `CF` to each sample. The definition is added to the header:

```
##FORMAT=<ID=CF,Number=.,Type=String,Description="Confidence class based on length␣
↪and copy ratio (HIGH, LOW)">
```

References

ensembl-vep. Sansa.

Part 3. Structural Variants Filtering and Secondary Annotations

Annotation Filtering and `SVTYPE` Selection

This multi-step workflow filters structural variant (SV) calls based on annotations (i.e, functional relevance, genomic location, allele frequency) and `SVTYPE`. It is mostly based on granite software. The output `vcf` file is checked for integrity to ensure the format is correct and the file is not truncated.

- CWL: workflow_granite-filtering_SV_selector_plus_vcf-integrity-check.cwl

Requirements

The input is a single annotated `vcf` file with the SV calls. The annotations must include genes and transcripts information (VEP) and allele frequency from gnomAD SV (Sansa).

Gene list

The gene list step uses granite to clean VEP annotations for transcripts that are not mapping to any gene of interest (not present in the CGAP Portal). This step does not remove any variants, but only modifies VEP annotations.

Inclusion List

The inclusion list step uses granite to filter-in exonic and functionally relevant variant based on VEP annotations. This step removes a large number of SVs from the initial call set.

Exclusion List

The exclusion list step uses granite to filter-out common variants based on gnomAD SV population allele frequency (`AF` > 0.01). Variants without gnomAD SV annotations are retained.

SV Type Selection

This step uses `SV_type_selector.py` script to filter out unwanted SV types. Currently only deletions (DEL) and duplications (DUP) are retained.

Output

The output is a filtered `vcf` file with fewer entries than the input `vcf` file. The content of the remaining entries is identical to the input (no information added or removed) minus the information removed by the gene list step.

20 Unrelated Filtering

This step uses `20_unrelated_SV_filter.py` script to identify common and artifactual SVs in 20 unrelated individuals and filter them out. SV calls for each of the 20 unrelated individuals were generated with Manta (see: https://cgap-annotations.readthedocs.io/en/latest/unrelated_references.html).

- CWL: workflow_20_unrelated_SV_filter_plus_vcf-integrity-check.cwl

Requirements

The input is a single annotated `vcf` file with the SV calls, alongside a `tar` archive of the `vcf` files with the SV calls for the 20 unrelated individuals. This step currently only work with DEL and DUP (which are provided to the `SVTYPE` argument), although the `vcf` files can contain other type of variants.

Matching and Filtering

When comparing SV calls from the input `vcf` file to the calls for an unrelated `vcf` file, the following logic applies to define a match:

1. `SVTYPE` must match
2. breakpoints at 5' end must be +/- 50 bp from each other
3. breakpoints at 3' end must be +/- 50 bp from each other

4. SVs must reciprocally overlap by a minimum of 80%

This produces a filtered `vcf` file that only contains SVs shared by a maximum of n individuals. The default is currently n = 1, such that SVs shared by 2 or more of the 20 unrelated individuals are filtered out.

Output

The output is a filtered `vcf` file containing fewer entries compared to the input `vcf` file. The variants that remain after filtering will receive an additional annotation, `UNRELATED=n`, where n is the number of matches found within the 20 unrelated SV calls.

Secondary Annotation

This workflow runs a series of scripts to add additional annotations to the SV calls in `vcf` format:

- `liftover_hg19.py` (https://github.com/dbmi-bgm/cgap-scripts) to add lift-over coordinates for breakpoint locations for **hg19/GRCh37** genome build
- `SV_worst_and_locations.py` to add information for breakpoint locations relative to impacted transcripts and the most severe consequence from VEP annotations
- `SV_cytoband.py` to add Cytoband information for the breakpoint locations

`SV_worst_and_locations.py` also implement some filtering and can result in fewer variants in the resulting `vcf` that is eventually checked for integrity.

Note: These scripts only work on DEL and DUP calls. Inversions (INV), break-end (BND), and insertions (INS) are not supported.

- CWL: workflow_SV_secondary_annotation_plus_vcf-integrity-check.cwl

Requirements

This workflow requires a single `vcf` file with the SV calls that went through **Annotation Filtering and** `SVTYPE` **Selection**. It also requires:

- The **hg38/GRCh38** to **hg19/GRCh37** chain file for lift-over (https://cgap-annotations.readthedocs.io/en/latest/liftover_chain_files.html#hg38-grch38-to-hg19-grch37)
- The **hg38/GRCh38** Cytoband reference file from UCSC (https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#cytoband)

Both the Cytoband annotation and the lift-over step require the `END` tag in the INFO field in the `vcf` file.

Annotation and Possible Filtering

1. For `liftover_hg19.py`, three lines are added to the header:

```
##INFO=<ID=hg19_chr,Number=.,Type=String,Description="CHROM in hg19 using LiftOver␣
↪from pyliftover">
##INFO=<ID=hg19_pos,Number=.,Type=Integer,Description="POS in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
##INFO=<ID=hg19_end,Number=1,Type=Integer,Description="END in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
```

The data associated with these tags are also added to the INFO field of the vcf file for qualifying variants using the following criteria:

- For the lift-over process to **hg19/GRCh37** coordinates, variants with successful conversions at both breakpoints will include data for the hg19_chr and both hg19_pos (breakpoint 1) and hg19_end (breakpoint 2) tags in the INFO field. If the conversion fails (e.g., if the coordinates do not have a corresponding location in **hg19/GRCh37**), the tags and any lift-over information will not be included in the output. Note that each breakpoint is treated separately, so it is possible for a variant to have data for hg19_chr and hg19_pos, but not hg19_end, or hg19_chr and hg19_end, but not hg19_pos
- Given that pyliftover does not convert ranges, the single-point coordinate in **hg38/GRCh38** corresponding to each variant CHROM and POS (or END) are used as query, and the **hg19/GRCh37** coordinate (result) will also be a single-point coordinate

2. For SV_worst_and_locations.py, three new fields are added to the CSQ tag in INFO field initially created by VEP. These are:

- Most_severe, which will have a value of 1 if the transcript is the most severe, and will otherwise be blank
- Variant_5_prime_location, which gives the location for breakpoint 1 relative to the transcript (options below)
- Variant_3_prime_location, which gives the location for breakpoint 2 relative to the transcript (options below)

Options for the location fields include: Indeterminate, Upstream, Downstream, Intronic, Exonic, 5_UTR, 3_UTR, Upstream_or_5_UTR, 3_UTR_or_Downstream, or Within_miRNA.

Additionally, for each variant this step removes annotated transcripts that do not possess one of the following biotypes: protein_coding, miRNA, or polymorphic_pseudogene. If after this cleaning a variant no longer has any annotated transcripts, that variant is also filtered out of the vcf file.

3. For SV_cytoband.py, the following two lines are added to the header:

```
##INFO=<ID=Cyto1,Number=1,Type=String,Description="Cytoband for SV start (POS) from␣
↪hg38 cytoBand.txt.gz from UCSC">
##INFO=<ID=Cyto2,Number=1,Type=String,Description="Cytoband for SV end (INFO END)␣
↪from hg38 cytoBand.txt.gz from UCSC">
```

Each variant will receive a Cyto1 annotation which corresponds to the Cytoband position of breakpoint 1 (which is POS in the vcf), and a Cyto2 annotation which corresponds to the Cytoband position of breakpoint 2 (which is END in the INFO field).

Output

The output is an annotated vcf file where secondary annotations are added to qualifying variants as described above. Some variants may be additionally filtered out as described.

Length Filtering

This step uses SV_length_filter.py to remove the largest SVs from the calls in the vcf file. The resulting vcf file is checked for integrity.

- CWL: workflow_SV_length_filter_plus_vcf-integrity-check.cwl

Requirements

This workflow requires a single `vcf` file with the SV calls and a parameter to define the maximum length allowed for the SVs.

Filtering

Currently we are filtering-out events larger than 10 Mb that we observed represent artifacts for the algorithm.

Output

This is the final `vcf` file that is ingested into the CGAP Portal.

VCF Annotation Cleaning

This step uses `SV_annotation_VCF_cleaner.py` script to remove most of VEP annotations to create a smaller `vcf` file for HiGlass visualization. This improves the loading speed in the genome browser. The resulting `vcf` file is checked for integrity.

- CWL: workflow_SV_annotation_VCF_cleaner_plus_vcf-integrity-check.cwl

Requirements

This workflow expects the final `vcf` file that is ingested into the CGAP Portal as input.

Cleaning

VEP annotations are removed from the `vcf` file and the REF and ALT fields are simplified using the `SV_annotation_VCF_cleaner.py` script.

Output

The output is a modified version of the final `vcf` file that is ingested into the CGAP Portal, that has been cleaned for the HiGlass genome browser. This file is also ingested into the CGAP Portal but only used for visualization.

References

granite.

References

Manta. ensembl-vep. Sansa. granite. VCFtools.

QC - SV Germline

This is a documentation for quality controls that are part of CGAP Pipelines module for germline Structural Variants (SVs).

VCF Quality Control

Overview

To evaluate the quality of a `vcf` file, different metrics are calculated using granite `SVqcVCF`.

The metrics currently available are:

- Variant types distribution per sample
- Total variant counts per sample

Definitions

Variant Types Distribution

Total number of variants classified by type as:

- **DEL**etion (`SVTYPE=DEL`)
- **DUP**lication (`SVTYPE=DUP`)
- Total variants (`SVTYPE=DEL` + `SVTYPE=DUP`)

Variants are only counted if the sample has a non-reference genotype (`0/1` or `1/1`).

References

granite.

News and Updates - SV Germline

Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v3 -> v1.0.0, we are starting a new more comprehensive versioning system

5.1.3 CGAP Pipeline - CNV Germline

This is the documentation for the CGAP Pipelines module for Copy Number Variants (CNVs) in germline data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-SV-germline.

Overview - CNV Germline

The CGAP Pipelines module for germline Copy Number Variants (CNVs) (https://github.com/dbmi-bgm/cgap-pipeline-SV-germline) identifies, annotates, and filters CNVs starting from analysis-ready `bam` files to produce final sets of calls in `vcf` format.

CNVs are a class of large genomic variants that include deletion and duplication events resulting in a copy number change. CNVs are a type of Structural Variants (SVs), which also includes inversions, translocations, and other complex genomic changes. However, it's useful to maintain a separate classification to account for the differences between the algorithms used for their detection. SV calling algorithms rely on local information from anomalously mapping reads (e.g., read pairs with unexpected fragment sizes, mapping orientations, and clipping events). CNV calling algorithms search instead for unexpected variation in sequencing coverage. As a result, algorithms for CNV detection perform better and are more robust in the identification of larger events, as they don't rely on local context information, but are less powerful and accurate than SV algorithms in the detection of smaller events. Given these substantial differences, CGAP implements both SV and a CNV calling algorithms, with the goal to combine the strength of both algorithmic approaches for an integrated analysis of the structural variation in the germline genome.

The pipeline is mostly based on the CNV calling algorithm BICseq2, alongside software for variants annotation and filtering (ensembl-vep, Sansa and granite).

The pipeline is designed for proband-only analysis, with the proband diagnosed with a likely monogenic disease. It can receive the initial analysis-ready `bam` file from either of the CGAP Upstream modules.

Note: The pipeline is not optimized for Whole Exome Sequencing (WES) data. Currently, the `bam` files used for input must be generated by mapping 150 bp paired-end reads to **hg38/GRCh38** genome assembly. The mappability file used by BICseq2 was generated on **hg38/GRCh38**, as well as the other reference files used in the annotation steps. The mappability file was calculated considering 150 bp reads.

Docker Images

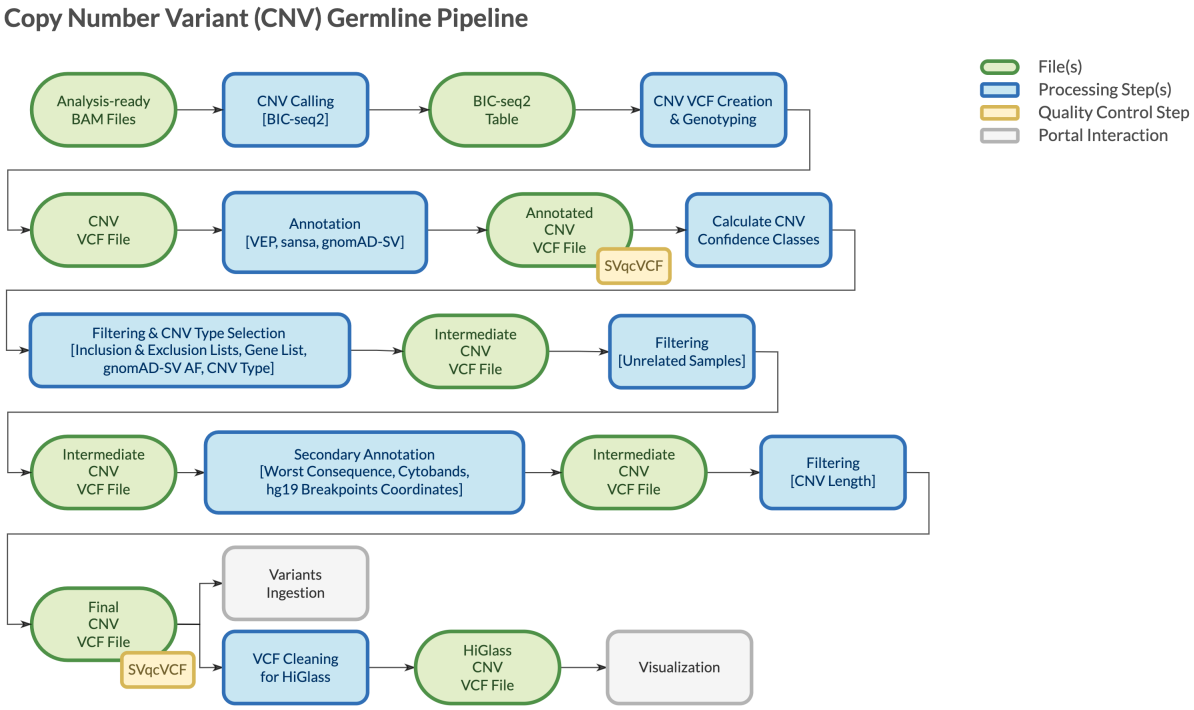
The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `cnv_germline` image is primarily for **CNV identification**. This image contains (but is not limited to) the following software packages:

- BICseq2 normalization (0.2.6)
- BICseq2 segmentation (0.7.3)
- R (4.1.0)
- granite (0.2.0)
- picard (2.26.11)
- samtools (1.9)

Pipeline Flow

The overall flow of the pipeline is shown below:



Pipeline Parts

Largely, the pipeline consists of three parts:

> Part 1. Starting from an analysis-ready `bam` file, BICseq2 algorithm identifies differences in coverage that are eventually converted to genotyped CNVs in `vcf` format using `bic_seq2_vcf_formatter.py` script
>
> Part 2. CNVs are annotated using VEP (Ensembl Variant Effect Predictor) to add transcripts information and Sansa to add gnomAD SV allele frequencies
>
> Part 3. CNVs are filtered to remove non-functional variants, artifacts, and variants that are common in the population

`vcf` files are checked for integrity using VCFtools `vcf-validator` at the end of each step during which they are created or modified.

Pipeline Steps

Part 1. BICseq2 CNVs Identification

BICseq2

This workflow runs BICseq2 algorithm to identify potential copy number variants (CNVs). It is divided in three parts, it starts from a `bam` file and produces a `txt` output table of genomic regions partitioned by observed and expected sequencing coverage.

- CWL: workflow_BICseq2_map_norm_seg.cwl

Input

The input is an analysis-ready `bam` file, generated from either of the CGAP Upstream modules. The `bam` file can also be provided by the user. Currently only 150 bp paired-end sequencing mapped to **hg38/GRCh38** is supported.

Generating BICseq2 `seq` Files

Reads position files (`seq`) are generated using `map.sh`. The script uses Samtools to filter reads based on final mapping quality score (MAPQ) and length. These `seq` files are the main input for the normalization step (`BICseq2-norm`). The fragment size for the library is also calculated using Samtools and Picard.

BICseq2-norm

Once the `seq` files are created, they can run through `BICseq2-norm` (normalization) using `norm.sh` script. The fragment size calculated for the `seq` files must be between 150 bp and 1500 bp. The workflow will fail if the fragment size is too large or too small.

It is important to note that BICseq2 is designed to work with a unique mappability file that is specific to a genome version and a read length. For information on how we generated our current mappability file (**hg38/GRCh38** and 150 bp), see: https://cgap-annotations.readthedocs.io/en/latest/bic-seq2_mappability.html.

With the appropriate mappability file, `BICseq2-norm` produces normalized sequencing coverage across the mappable genome. Given the expectation for diploid genome, BICseq2 algorithm can only apply to autosomes.

BICseq2-seg

`BICseq2-seg` (segmentation) partitions the genome into regions based on the number of observed and expected reads produced by the normalization step. The output is a `txt` table with genomic regions, the number of observed and expected reads within those regions, the *log2.copyRatio* between observed and expected reads, and *pvalues* that indicate how significant the *log2.copyRatio* change is from the expected null of 0.

Reformatting BICseq2 Output to `vcf`

This workflow uses `bic_seq2_vcf_formatter.py` script to filter the output table from BICseq2 and convert the CNVs calls in `vcf` format, adding `SVTYPE` (DEL or DUP) and genotype information. The resulting `vcf` file is checked for integrity.

- CWL: workflow_BICseq2_to_vcf_plus_vcf-integrity-check.cwl

Calling DEL or DUP

The following logic is used to classify BICseq2 calls as deletions (DEL) or duplications (DUP):

1. `'-p','--pvalue'`: the *pvalue* below which a region could be called as a deletions or duplications; currently 0.05
2. `'-d','--log2_min_dup'`: positive value (*log2.copyRatio*) above which a duplication is called; currently 0.2
3. `'-e','--log2_min_del'`: negative value (*log2.copyRatio*) below which a deletion is called; currently (-)0.2

A DEL is called if `pvalue` and `log2_min_del` are both true. A DUP is called if `pvalue` and `log2_min_dup` are both true. If only one of the `pvalue` or the `log2_min` for the variant type is true, the region is not reported.

Genotyping DEL or DUP

If a variant qualifies as a DEL or DUP, it must be genotyped. The following logic applies:

1. `'-u','--log2_min_hom_dup'`: positive value (*log2.copyRatio*) above which a homozygous or high copy number duplication is called; currently 0.8
2. `'-l','--log2_min_hom_del'`: negative value (*log2.copyRatio*) below which a homozygous deletion is called; currently (-)3.0

If a DEL is valid, but displays a *log2.copyRatio* between `log2_min_del` and `log2_min_hom_del`, it will be genotyped as heterozygous `0/1`.

If a DUP is valid, but displays a *log2.copyRatio* between `log2_min_dup` and `log2_min_hom_dup`, it will be genotyped as heterozygous `0/1`.

We currently do not provide a true genotype for DUPs with *log2.copyRatio* > `log2_min_hom_dup`, because unlike DELs, proband-only CNV analysis cannot conclude the phase of duplications. For example, with two extra copies, each parent could provide one copy `1/1` or one parent could provide two copies `2/0`. In this case we will genotype the variant as unknown `./.`.

References

BICseq2.

Part 2. CNVs Annotation

Sansa and VEP

This workflow uses Sansa and VEP (Ensembl Variant Effect Predictor) to annotate the identified copy number variants (CNVs) and combines the annotations into a single `vcf` file. The resulting `vcf` file is checked for integrity.

Note: The gnomAD SV database, which we annotate against, only contains calls from SV algorithms. This is currently the best available dataset for SVs, but does not contain calls generated by CNV algorithms.

- CWL: workflow_sansa_vep_combined_annotation_plus_vcf-integrity-check.cwl

Sansa

`sansa.sh` script executes Sansa to identify CNVs that match the **hg38/GRCh38** lift-over of the gnomAD v2 structural variants database (https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#gnomad-structural-variants). The script sorts the input `vcf` file and runs the following command for Sansa:

```
sansa annotate -m -n -b 50 -r 0.8 -s all -d $gnomAD $vcf
```

- `-m` returns all CNVs (including those without matches to the database)
- `-n` allows for matches between different SV types (to allow CNV to match DEL and DUP)
- `-b 50` is the default parameter for maximum breakpoint offset (in bp) between the newly-identified CNV and the SV in gnomAD SV
- `-r 0.8` is nearly the default parameter (default is 0.800000012) for minimum reciprocal overlap between SVs

- `-s all` provides all matches instead of automatically selecting the single best match
- `-d $gnomAD` is the gnomAD SV database
- `$vcf` is the input `vcf` file

VEP

`vep-annot_SV.sh` script executes VEP to annotate the CNVs with genes and transcripts. The script produces an annotated `vcf` containing all variants.

A maximum CNV size slightly larger than *chr1* in the **hg38/GRCh38** genome (`--max_sv_size 250000000`) is used in the VEP command in order to avoid filtering of large CNVs. The `--overlaps` option is also included to record the overlap between the VEP features and the CNVs (reported in bp and percentage). The `--canonical` option is included to flag canonical transcripts.

Combine Sansa and VEP

The outputs from Sansa and VEP are combined using `combine_sansa_and_VEP_vcf.py` script. The `vcf` file generated by VEP is used as a scaffold, onto which gnomAD SV annotations from Sansa are added.

When multiple matches are identified in the gnomAD SV database, the following logic applies to select the best (and rarest) match:

1. Select a type-matched CNV (if possible), and the rarest type-matched variant from gnomAD SV (using `AF`) if there are multiple matches
2. If none of the options are a type-match, select the rarest variant from gnomAD SV (using `AF`)

Note: CNV is a variant class in gnomAD SV, but not in the BICseq2 output. Since DELs and DUPs are types of CNVs, we prioritize as follows: (I) we first search for type-matches between DEL and DEL or DUP and DUP, (II) if a type-match is not found for the variant, we then search for type-matches between DEL and CNV or DUP and CNV, (III) all other combinations (e.g., INV and CNV, or DEL and DUP) are considered to **not** be type-matched.

These rules were set given limitations on the number of values the gnomAD SV fields can have for filtering in the CGAP Portal and to avoid loss of rare variants in the upcoming filtering steps. The final output is a `vcf` file with annotations for both gene/transcript and gnomAD SV population frequencies.

Confidence Classes

This workflow assigns a confidence class to each of the CNVs identified by the pipeline.

- CWL: BICseq2_add_confidence.cwl

Confidence classes are calculated and assigned using the `SV_confidence.py` script. A single `vcf` is required as input, and the file must contain the information supporting each of the calls created by BICseq2. The possible confidence classes are:

- HIGH
- LOW

The confidence classes are calculated based on the following parameters:

- *length*: the length of the variant, calculated as the absolute value of the `SVLEN` field
- *log-ratio*: the BICseq2_log2_copyRatio parameter calculated by BICseq2

Each variant is classified based on the following criteria:

High Confidence Calls

```
length > 1 Mbp & (log-ratio > 0.4 || log-ratio < -0.8)
```

Low Confidence Calls

All the other variants.

The calculated confidence classes are added as the new FORMAT field `CF` to the sample:

```
##FORMAT=<ID=CF,Number=.,Type=String,Description="Confidence class based on length␣
↪and copy ratio (HIGH, LOW)">
```

References

ensembl-vep. Sansa.

Part 3. CNVs Filtering and Secondary Annotations

Annotation Filtering and `SVTYPE` Selection

This multi-step workflow filters copy number variant (CNV) calls based on annotations (i.e, functional relevance, genomic location, allele frequency) and `SVTYPE`. It is mostly based on granite software. The output `vcf` file is checked for integrity to ensure the format is correct and the file is not truncated.

- CWL: workflow_granite-filtering_SV_selector_plus_vcf-integrity-check.cwl

Requirements

The input is a single annotated `vcf` file with the CNV calls. The annotations must include genes and transcripts information (VEP) and allele frequency from gnomAD SV (Sansa).

Gene list

The gene list step uses granite to clean VEP annotations for transcripts that are not mapping to any gene of interest (not present in the CGAP Portal). This step does not remove any variants, but only modifies VEP annotations.

Inclusion List

The inclusion list step uses granite to filter-in exonic and functionally relevant variant based on VEP annotations. This step removes a large number of CNVs from the initial call set.

Exclusion List

The exclusion list step uses granite to filter-out common variants based on gnomAD SV population allele frequency (`AF` > 0.01). Variants without gnomAD SV annotations are retained.

CNV Type Selection

This step uses `SV_type_selector.py` script to filter out unwanted CNV types. Currently we are not filtering any of the possible CNV types as we retain both deletions (DEL) and duplications (DUP).

Output

The output is a filtered `vcf` file with fewer entries than the input `vcf` file. The content of the remaining entries is identical to the input (no information added or removed) minus the information removed by the gene list step.

20 Unrelated Filtering

This step uses `20_unrelated_SV_filter.py` script to identify common and artifactual CNVs in 20 unrelated individuals and filter them out. CNV calls for each of the 20 unrelated individuals were generated with BICseq2 (see: https://cgap-annotations.readthedocs.io/en/latest/unrelated_references.html).

- CWL: workflow_20_unrelated_SV_filter_plus_vcf-integrity-check.cwl

Requirements

The input is a single annotated `vcf` file with the CNV calls, alongside a `tar` archive of the `vcf` files with the CNV calls for the 20 unrelated individuals. This step currently work with both DEL and DUP (which are provided to the `SVTYPE` argument), accounting for all possible CNV types.

Matching and Filtering

When comparing CNV calls from the input `vcf` file to the calls for an unrelated `vcf` file, the following logic applies to define a match:

1. `SVTYPE` must match
2. breakpoints at 5' end must be +/- 50 bp from each other
3. breakpoints at 3' end must be +/- 50 bp from each other
4. CNVs must reciprocally overlap by a minimum of 80%

This produces a filtered `vcf` file that only contains CNVs shared by a maximum of n individuals. The default is currently n = 1, such that CNVs shared by 2 or more of the 20 unrelated individuals are filtered out.

Output

The output is a filtered `vcf` file containing fewer entries compared to the input `vcf` file. The variants that remain after filtering will receive an additional annotation, `UNRELATED=n`, where n is the number of matches found within the 20 unrelated CNV calls.

Secondary Annotation

This workflow runs a series of scripts to add additional annotations to the CNV calls in `vcf` format:

- `liftover_hg19.py` (https://github.com/dbmi-bgm/cgap-scripts) to add lift-over coordinates for breakpoint locations for **hg19/GRCh37** genome build
- `SV_worst_and_locations.py` to add information for breakpoint locations relative to impacted transcripts and the most severe consequence from VEP annotations
- `SV_cytoband.py` to add Cytoband information for the breakpoint locations

`SV_worst_and_locations.py` also implement some filtering and can result in fewer variants in the resulting `vcf` that is eventually checked for integrity.

Note: These scripts only work on DEL and DUP calls. Inversions (INV), break-end (BND), and insertions (INS) are not supported.

- CWL: workflow_SV_secondary_annotation_plus_vcf-integrity-check.cwl

Requirements

This workflow requires a single `vcf` file with the CNV calls that went through **Annotation Filtering and** `SVTYPE` **Selection**. It also requires:

- The **hg38/GRCh38** to **hg19/GRCh37** chain file for lift-over (https://cgap-annotations.readthedocs.io/en/latest/liftover_chain_files.html#hg38-grch38-to-hg19-grch37)
- The **hg38/GRCh38** Cytoband reference file from UCSC (https://cgap-annotations.readthedocs.io/en/latest/variants_sources.html#cytoband)

Both the Cytoband annotation and the lift-over step require the `END` tag in the INFO field in the `vcf` file.

Annotation and Possible Filtering

1. For `liftover_hg19.py`, three lines are added to the header:

```
##INFO=<ID=hg19_chr,Number=.,Type=String,Description="CHROM in hg19 using LiftOver␣
↪from pyliftover">
##INFO=<ID=hg19_pos,Number=.,Type=Integer,Description="POS in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
##INFO=<ID=hg19_end,Number=1,Type=Integer,Description="END in hg19 using LiftOver␣
↪from pyliftover (converted back to 1-based)">
```

The data associated with these tags are also added to the INFO field of the `vcf` file for qualifying variants using the following criteria:

- For the lift-over process to **hg19/GRCh37** coordinates, variants with successful conversions at both breakpoints will include data for the `hg19_chr` and both `hg19_pos` (breakpoint 1) and `hg19_end` (breakpoint 2) tags in the INFO field. If the conversion fails (e.g., if the coordinates do not have a corresponding location in **hg19/GRCh37**), the tags and any lift-over information will not be included in the output. Note that each breakpoint is treated separately, so it is possible for a variant to have data for `hg19_chr` and `hg19_pos`, but not `hg19_end`, or `hg19_chr` and `hg19_end`, but not `hg19_pos`
- Given that pyliftover does not convert ranges, the single-point coordinate in **hg38/GRCh38** corresponding to each variant CHROM and POS (or `END`) are used as query, and the **hg19/GRCh37** coordinate (result) will also be a single-point coordinate

2. For `SV_worst_and_locations.py`, three new fields are added to the `CSQ` tag in INFO field initially created by VEP. These are:

- `Most_severe`, which will have a value of `1` if the transcript is the most severe, and will otherwise be blank

- `Variant_5_prime_location`, which gives the location for breakpoint 1 relative to the transcript (options below)
- `Variant_3_prime_location`, which gives the location for breakpoint 2 relative to the transcript (options below)

Options for the location fields include: `Indeterminate`, `Upstream`, `Downstream`, `Intronic`, `Exonic`, `5_UTR`, `3_UTR`, `Upstream_or_5_UTR`, `3_UTR_or_Downstream`, or `Within_miRNA`.

Additionally, for each variant this step removes annotated transcripts that do not possess one of the following biotypes: `protein_coding`, `miRNA`, or `polymorphic_pseudogene`. If after this cleaning a variant no longer has any annotated transcripts, that variant is also filtered out of the `vcf` file.

3. For `SV_cytoband.py`, the following two lines are added to the header:

```
##INFO=<ID=Cyto1,Number=1,Type=String,Description="Cytoband for SV start (POS) from␣
↪hg38 cytoBand.txt.gz from UCSC">
##INFO=<ID=Cyto2,Number=1,Type=String,Description="Cytoband for SV end (INFO END)␣
↪from hg38 cytoBand.txt.gz from UCSC">
```

Each variant will receive a `Cyto1` annotation which corresponds to the Cytoband position of breakpoint 1 (which is POS in the `vcf`), and a `Cyto2` annotation which corresponds to the Cytoband position of breakpoint 2 (which is `END` in the INFO field).

Output

The output is an annotated `vcf` file where secondary annotations are added to qualifying variants as described above. Some variants may be additionally filtered out as described.

Length Filtering

Note: We are NOT currently length filtering BICseq2 CNV calls. The workflow is turned off by specifying a maximum length argument that is larger than chr1 (250000000 bp), the same value used to run VEP.

This step uses `SV_length_filter.py` to remove the largest CNVs from the calls in the `vcf` file. The resulting `vcf` file is checked for integrity.

- CWL: workflow_SV_length_filter_plus_vcf-integrity-check.cwl

Requirements

This workflow requires a single `vcf` file with the CNV calls and a parameter to define the maximum length allowed for the SVs.

Filtering

Currently none.

Output

This is the final `vcf` file that is ingested into the CGAP Portal.

VCF Annotation Cleaning

This step uses `SV_annotation_VCF_cleaner.py` script to remove most of VEP annotations to create a smaller `vcf` file for HiGlass visualization. This improves the loading speed in the genome browser. The resulting `vcf` file is checked for integrity.

- CWL: workflow_SV_annotation_VCF_cleaner_plus_vcf-integrity-check.cwl

Requirements

This workflow expects the final `vcf` file that is ingested into the CGAP Portal as input.

Cleaning

VEP annotations are removed from the `vcf` file and the REF and ALT fields are simplified using the `SV_annotation_VCF_cleaner.py` script.

Output

The output is a modified version of the final `vcf` file that is ingested into the CGAP Portal, that has been cleaned for the HiGlass genome browser. This file is also ingested into the CGAP Portal but only used for visualization.

References

granite.

References

BICseq2. ensembl-vep. Sansa. granite. VCFtools.

News and Updates - CNV Germline

Version Updates

v1.1.0

- Conversion to YAML format for portal objects
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- v3 -> v1.0.0, we are starting a new more comprehensive versioning system

5.2 Somatic

Somatic pipelines are designed to identify somatic mutations, which are changes that occur in an individual's DNA during their lifetime and are present in certain cells only. These mutations are often associated with cancer and can be identified using matched tumor-normal samples or tumor-only data in order to discover mutations that may be related to cancer development or progression.

5.2.1 CGAP Pipeline - Somatic Sentieon

This is the documentation for the CGAP Pipelines module for variant calling with Sentieon in somatic data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-somatic-sentieon.

Overview - Somatic Sentieon

The CGAP Pipelines module for somatic variant calling with Sentieon (https://github.com/dbmi-bgm/cgap-pipeline-somatic-sentieon) is our *license-based* option for calling Single Nucleotide Variants (SNVs), short Insertions and Deletions (INDELs), and Structural Variants (SVs) for Whole Genome Sequencing (WGS) Tumor-Normal paired data. The pipeline starts from matching analysis-ready `bam` files for a Tumor and a corresponding Normal (non-Tumor) tissue for the same individual. It can receive the initial `bam` files from either of the CGAP Upstream modules. The output of the pipeline is a `vcf` file with the variant calls that are unique of the Tumor.

Note: If the user is providing `bam` files as input, the files must be aligned to **hg38/GRCh38** for compatibility with the annotation steps.

Docker Image

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The image contains (but is not limited to) the following software packages:

- Sentieon (202112.01)
- samtools (1.9)

Pipeline Flow

Our implementation offers a one step end-to-end solution to run a Tumor-Normal analysis using the Sentieon `TNscope` algorithm as described here. We are using of a Panel of Normal (PON) `vcf` file generated from 20 unrelated samples from The Utah Genome Project (UGRP) as described here (https://cgap-annotations.readthedocs.io/en/latest/unrelated_references.html).

References

Sentieon.

News and Updates - Somatic Sentieon

Version Updates

v1.1.0

- Conversion to YAML standard
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- Initial release

5.2.2 CGAP Pipeline - SNV Somatic

This is the documentation for the CGAP Pipelines module for Single Nucleotide Variants (SNVs) in somatic data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-SNV-somatic.

Overview - SNV Somatic

The CGAP Pipelines module for somatic Single Nucleotide Variants (SNVs) (https://github.com/dbmi-bgm/cgap-pipeline-SNV-somatic) is our solution to filter and annotate the variants called by CGAP Somatic Sentieon module. The input `vcf` contains SNVs, short Insertions and Deletions (INDELs), and Structural Variants (SVs), which are filtered, annotated and reformatted for use in the somatic browser.

Docker Image

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The image contains (but is not limited to) the following software packages:

- vcftools (954e607)
- granite (0.2.0)

Pipeline Steps

Part 1. Filtering and Splitting the VCF

In development.

References

Sentieon.

News and Updates - SNV Somatic

Version Updates

v1.1.0

- Conversion to YAML standard
- Added components to prioritize driver mutations using Hartwig decision tree
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- Initial release

5.2.3 CGAP Pipeline - CNV Somatic

This is the documentation for the CGAP Pipelines module for Copy Number Variants (CNVs) in somatic data. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-SV-somatic.

Overview - CNV Somatic

The CGAP Pipelines module for somatic Copy Number Variants (CNVs) (https://github.com/dbmi-bgm/cgap-pipeline-SV-somatic) identifies, annotates, and filters CNVs starting from analysis-ready `bam` files to produce final sets of calls in `vcf` format.

CNVs are a class of large genomic variants that result in a change in copy number, including deletions (also referred to as losses) and duplications (also referred to as gains). CNVs are identified by algorithms that search for unexpected differences in sequencing coverage.

The pipeline is mostly based on the CNV calling algorithm ASCAT, and calls variants from Whole Genome Sequencing (WGS) Tumor-Normal paired samples. It can receive the initial analysis-ready `bam` files from either of the CGAP Upstream modules.

Note: If the user is providing `bam` files as input, the files must be aligned to **hg38/GRCh38** for compatibility with the annotation steps.

Docker Image

The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The `ascat` image is primarily for **CNV identification**. This image contains (but is not limited to) the following software packages:

- R (4.1.0)
- ascat (3.0.0)
- alleleCount (4.3.0)

Pipeline Steps

ASCAT

This step calls copy number variants (CNVs) in Tumor-Normal paired samples with the ASCAT algorithm using `ascat.R` script.

- CWL: ascat.cwl

Input

The user should provide the following files and parameters:

- Analysis-ready `bam` file containing aligned sequencing reads for a Tumor sample
- Analysis-ready `bam` file containing aligned sequencing reads for a Normal sample
- `gender` parameter for the gender information, accepted values are XX (female) or XY (male)
- `nthreads` parameter for the number of threads to run ASCAT; default value is 23

The algorithm also requires additional reference files that include loci, allele and, GC correction files. For more details on these files, see (https://cgap-annotations.readthedocs.io/en/latest/ascat.html).

Running ASCAT

`run_ascat.sh` calls `ascat.R` to run the CNV analysis.

The major steps of the `ASCAT` algorithm are:

1. Calculate allele counts and allele fractions
   - Runs `alleleCount` to collect allele counts at specific loci for Normal and Tumor samples
   - Obtains B-allele Fraction (BAF) and LogR from the raw allele counts
2. Plot raw data
   - Generates `png` files presenting the BAF and LogR for the Normal and Tumor samples
3. Correct LogR
   - Corrects LogR of the Tumor sample with genomic GC content
4. Plot corrected data
   - Generates `png` files presenting BAF and LogR for the Normal and Tumor samples after the GC correction
5. Run Allele Specific Piecewise Constant Fitting (ASPCF)
   - It is a preprocessing step that fits piecewise constant regression to both the LogR and the BAF data at the same time. This method identifies regions (segments), which are genomics regions between two consecutive change points. Each segment has assigned a single LogR value and either one or two BAF values (a single BAF equal to 0.5 is assigned if the aberrant cells are discovered as balanced). The segmented data are saved to a `png` file. After this operation, partial results are saved to a `tsv` file containing the information about: LogR and BAF for both germline and somatic samples, LogR and BAF segmented for the Tumor sample.
6. Run allele specific copy number analysis of tumors

- Determines estimated values of aberrant cell fractions, tumor ploidy, and allele specific copy number calls. Minor and major copy numbers for the segments are obtained. The results of this step are saved to a `tsv` file that contains start and end positions of the segments with the assigned minor and major copy numbers and their sum. A plot presenting ASCAT profiles of the sample is saved to a `png` file, and a plot showing ASCAT raw profiles. ASCAT evaluates the ploidy of the tumor cells and the fraction of abberant cells considering all their possible values, and finally selects the optimal solution. A graphical representation of these values is saved to a `png` file.

Others

In order to reproduce the obtained results, some of the ASCAT objects are saved in an `Rdata` file, which stores the following objects:

- `ascat.bc` - an object returned from the `ascat.aspcf` function
- `ascat.output` - an object returned from the `ascat.RunAscat` function
- `QC` - an object that stores various ascat metrics returned from the `ascat.metrics` function

References

ASCAT.

References

ASCAT.

News and Updates - CNV Somatic

Version Updates

v1.1.0

- Conversion to YAML standard
- Added components to prioritize driver mutations using Hartwig decision tree
- *FileReference* objects shared by multiple pipelines have been centralized in Base

v1.0.0

- Initial release

CHAPTER 6

Other Pipelines

The CGAP Pipelines are modular and organized in a repository structure that enables open-source contribution and the streamlined integration of new algorithms and pipelines. These additions can build upon our current offerings starting from the final output or intermediate files of our existing pipelines and adding additional analyses. This flexible approach allows for continuous improvement and expansion of the capabilities of CGAP Pipelines.

6.1 CGAP Pipeline - Sentieon Joint Calling

This is the documentation for the CGAP Pipelines module for joint calling and genotyping with Sentieon. The pipeline components are bundled in the GitHub repository https://github.com/dbmi-bgm/cgap-pipeline-upstream-sentieon.

6.1.1 Overview - Sentieon Joint Calling

The CGAP Pipelines module for joint calling and genotyping with Sentieon (https://github.com/dbmi-bgm/cgap-pipeline-upstream-sentieon) accepts multiple individual `g.vcf` files and produces a jointly genotyped `vcf` file as output. The `g.vcf` files can be generated through standard CGAP Pipelines processing (either of the CGAP Upstream modules followed by HaplotypeCaller) or can be provided by the user.

This pipeline is based on Sentieon and the `GVCFtyper` algorithm is used to combine the `g.vcf` files and joint genotype the resulting variants.

Docker Image

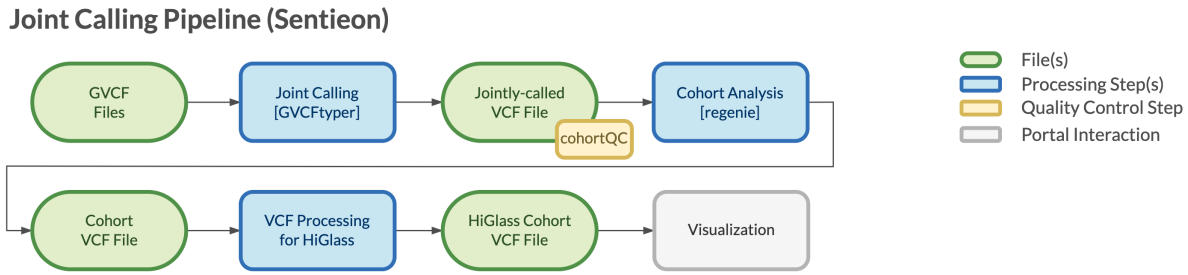
The Dockerfiles provided in this GitHub repository can be used to build public docker images. If built through `portal-pipeline-utils pipeline_deploy` command (https://github.com/dbmi-bgm/portal-pipeline-utils), private ECR images will be created for the target AWS account.

The image contains (but is not limited to) the following software packages:

- sentieon (202112.01)

Pipeline Flow

The overall flow of the pipeline is shown below:



Pipeline Steps

GVCFtyper

This step uses Sentieon `GVCFtyper` algorithm to jointly call and genotype single nucleotide variants (SNVs) and small INDELs.

- CWL: sentieon-GVCFtyper.cwl

Requirements

Requires input file(s) in `g.vcf` format generated through GATK `HaplotypeCaller` algorithm.

Parameters

To mirror our SNV Germline processing, which uses a `--standard-min-confidence-threshold-for-calling` default of `10` in the GATK `GenotypeGVCFs` step, we set the following parameters to run Sentieon `GVCFtyper`.

1. `--call_conf` is set to `10`
2. `--emit_conf` is set to `10`
3. `--emit_mode` is set to `variant`

Output

This step creates an output `vcf` file that stores jointly genotyped variants for all samples that are called together.

References

Sentieon GVCFtyper. GATK HaplotypeCaller. GATK GenotypeGVCFs.

References

Sentieon.

6.1.2 News and Updates - Sentieon Joint Calling

Version Updates

v1.1.0

- Conversion to YAML standard

v1.0.0

- Initial release

CHAPTER 7

Support Repositories

Together with the main repositories for the pipeline modules, additional repositories are used to store components to create annotation and reference files, and deploy the pipelines in CGAP infrastructure. Dedicated documentation is also available.

7.1 Portal Pipeline Utils

Pipelines can be automatically deployed to a target AWS account. More information is available in the documentation for pipeline utilities. The source code is stored in in the GitHub repository https://github.com/dbmi-bgm/portal-pipeline-utils.

7.2 CGAP Annotations and Reference Files

Detailed information on the data sources used for annotation and the reference files used by the pipelines is provided here. If pre-processing is required, the scripts to pre-process the raw data are available here.

7.3 CGAP Scripts

The pipelines use general purpose scripts that are shared by multiple steps. These scripts are stored in the GitHub repository https://github.com/dbmi-bgm/cgap-scripts.