54gene-wgs-germline

Bari Jane Ballew, Esha Joshi, Cameron Palmer

Aug 10, 2022

CONTENTS

CHAPTER

ONE

DOCUMENTATION OF WGS VARIANT CALLING AT 54GENE

1.1 Overview

This workflow was designed by the Genomics & Data Science team (GDS) at 54gene and is used to analyze paired-end short-read germline whole-genome sequencing data. This pipeline is designed to first be deployed in small batches (e.g. per flow cell), starting with FASTQs and resulting in gVCFs and a small batch joint-called VCF. A second run of the pipeline can receive a larger batch of gVCFs (e.g. gVCFs derived from many flow cells), and generates a large batch joint-called VCF. The workflow, which is designed to support reproducible bioinformatics, is written in Snakemake and is platform-agnostic. All dependencies are installed by the pipeline as-needed using conda. Development and testing has been predominantly on AWS' ParallelCluster using Amazon Linux using Snakemake version 7.8.2.

Features:

- Read filtering and trimming
- Read alignment, deduplication, and BQSR
- Variant calling and filtering
- Joint-genotyping
- Sex discordance and relatedness assessment
- Generate MultiQC reports

To install the latest release, type:

```
git clone https://gitlab.com/data-analysis5/dna-sequencing/54gene-wgs-germline.git
```

1.1.1 Inputs

The pipeline requires the following inputs:

- A headerless, whitespace delimited `manifest.txt` file with sample names and paths (columns dependent on the run-mode)
- Config file with the run-mode specified and other pipeline parameters configured (see default config provided in `config/config.yaml`)
- A tab-delimited `intervals.tsv` file with names of intervals and paths to region (BED) files of the genome you want to parallelize the variant calling and joint-calling steps by (i.e. 50 BED files each with a small region of the genome to parallelize by)
- A tab-delimited `sex_linker.tsv` file with the sample names in one column and sex in the other to identify discordances in reported vs. inferred sex

- A `multiqc.yaml` config file for generating MultiQC reports (provided for you)

1.1.2 Outputs

Depending on which run-mode you have set, you will be able to generate:

- A hard-filtered, multi-sample joint-called VCF in `full` and `joint_genotyping` mode
- Per-sample gVCFs for all regions of the genome for future joint-calling in `full` mode
- Deduplicated and post-BQSR BAM files in `full` mode
- Various QC metrics (e.g. FastQC, MultiQC, bcftools stats) in all three modes

See the *Installation*, *Execution*, and *Configuration* for details on setting up and running the pipeline.

1.2 Installation

This workflow was designed to use conda for dependency management and utilizes Snakemake as the workflow management system, ensuring reproducible and scalable analyses.

This installation guide is designed for Unix/Linux environments; however, this pipeline has been minimally tested on OSX as well.

1.2.1 Obtain a copy of this workflow

Clone this repository to your local system, into the place where you want to perform the data analysis:

```
git clone git@gitlab.com:data-analysis5/54gene-wgs-germline.git
```

1.2.2 Install the run-time environment

If needed, follow this guide to install Miniconda.

Once installed, create the run-time conda environment with minimal dependencies defined using the following command:

```
conda env create -f environment.yaml
```

Activate the environment as follows:

```
conda activate 54gene-wgs-germline
```

1.3 Configuration

The workflow needs to be configured to perform the analysis of your choice by editing the following files in the `config/` folder. Each file is described in more detail below.

1.3.1 Configuration file

The pipeline offers three run modes. Please specify the run mode in `config.yaml`. The name of the file defaults to `config.yaml` but you can use other filenames in conjunction with Snakemake's `--configfile` command line flag.

- **full**: This mode starts with FASTQs and emits a joint-called, filtered, multi-sample VCF.
- **joint_genotyping**: This mode starts with gVCFs and runs joint-calling and filtering, emitting a multi-sample VCF. In the event you have analyzed batches of samples in the full-run mode, these batches can then jointly re-genotyped with this run mode.
- **fastqc_only**: This mode starts with FASTQs and emits trimmed FASTQs as well as a multiQC report for raw and trimmed reads. This run mode is meant for performing QC on FASTQ data before further downstream analysis.

1.3.2 Manifest file

You will need to provide a headerless, white-space delimited manifest file to run the pipeline for all three run-modes. The default name for the file is `manifest.txt` but this is user configurable in the config file under `sampleFile`.

For **full** and **fastqc_only** mode, the `manifest.txt` requires the following columns:

- Columns: `readgroup sample_ID path/to/r1.fastq path/to/r2.fastq`
- `readgroup` values should be unique, e.g. <sampleID>_<barcode>_<lane>
- `sample_ID` should be the same for all FASTQ pairs from a single sample, and can be different from the FASTQ filenames

For example:

```
Sample1_S1_L001 Sample1 input/Sample_001_S1_L001_R1.fastq   input/Sample_001_S1_L001_R2.
↪fastq
Sample1_S1_L002 Sample1 input/Sample_001_S1_L002_R1.fastq   input/Sample_001_S1_L002_R2.
↪fastq
```

For **joint_genotyping** mode:

- Columns: `sample_ID path/to/file.g.vcf.gz`
- `sample_ID` values should be unique, and should correspond to the sample IDs in the gVCF header
- gVCFs should be bgzipped and indexed

For example:

```
Sample1 vcfs/Sample1.g.vcf.gz
Sample2 vcfs/Sample2.g.vcf.gz
```

1.3.3 Intervals file

For **full** and **joint_genotyping** modes only.

Joint-calling for a large number of samples is computationally expensive and time-consuming. This pipeline was designed to mitigate these issues by parallelizing joint-calling over multiple intervals of the genome. To specify the number of intervals, and which regions to parallelize over, a 2-column tab-delmited `intervals.tsv` file can be specified. The filename can be customized and edited in the config file under `intervalsFile`.

This file contains two columns with headers:

- `interval_name` for the name of the particular interval or region
- `file_path` full path to the interval/region BED file, Picard-style `.interval_list`, VCF file, or GATK-style `.list` or `.intervals` file (see further details on these formats here)

For example:

```
interval_name   file_path
interval_1  resources/scattered_calling_intervals/interval_1.bed
interval_2  resources/scattered_calling_intervals/interval_2.bed
```

The pipeline will supply these interval files to the GATK `HaplotypeCaller`, `GenomicsDBImport`, and `GenotypeGVCFs` steps to run concurrent instances of these rules at each specified interval(s), reducing overall execution time.

We recommend specifying regions of equal size for parallelization.

1.3.4 Sex linker file

The pipeline provides a boolean option `somalier` to estimate relatedness amongst the samples using Somalier in the `config.yaml` (see `check_relatedness` parameter in *Configuration*). This requires a 2-column, tab-delimited file. The filename defaults to `sex_linker.tsv` and is specified in the `config.yaml` under `sexLinker`. This file requires:

- First column with the header `Sample` with all sample names
- Second column with the header `Sex` containing case-insensitive encodings of sex in either m/f or male/female format

For example:

```
Sample  Sex
NA12878 F
Subject1    female
Subject2    m
```

1.3.5 MultiQC configuration

A configuration file for MultiQC can be found in `config/multiqc.yaml` and is used for generating and specifying the order of the various modules in the MultiQC report from the pipeline. We **do not** recommend modifying this file unless you understand how this configuration file is setup or how MultiQC works.

1.3.6 Config parameters

Below are descriptions and usage options for the various config parameters specified in `config.yaml`.

Parameter	Required	Description
`sampleFile`	Y	Manifest file with IDs
`intervals`	Y	File with interval names and file paths
`jobs`	Y	Max jobs to run concurrently
`sexLinker`	Y	File with reported sex of each sample ID
`tempDir`	Y	Location of temp directory; does not have to exist prior to pipeline execution
`runType`	Y	Specify run mode to use (see below)
`full`	Y	`[yes|no]` Set to yes for full run mode
`joint_genotyping`	Y	`[yes|no]` Set to yes for joint calling from gVCFs
`fastq_qc_only`	Y	`[yes|no]` Set to yes for FASTQ QC and trimming
`global_vars`	Y	Set global java options
`cluster_mode`	Y	Used to submit jobs to a cluster only if you are using the optional wrapper script. See *Execution*
`default_queue`	Y	Name of your default cluster partition/queue; can be ~
`compute_queue`	Y	Name of queue/partition best suited for compute- intensive jobs; can be ~
`memory_queue`	Y	Name of queue/partition best suited for memory-intensive jobs; can be ~
`center_id`	Y	Name of sequencing center for use in `@RG` tag in bams
`max_concurrent`	Y	Max concurrent jobs for specific high-bandwidth rules, to avoid potentially hitting bandwidth caps if deployed in a cloud environment; see wrapper script for an example of how to pass this in to snakemake. Set to the same number as `jobs` if you don't want to limit concurrent rules in this way
`max_het_ratio`	Y	Max het/hom ratio to allow through post-calling QC
`min_avg_depth`	Y	Minimum depth required for sample to pass post-calling QC
`max_contam`	Y	Max % contamination to allow through post-calling QC
`time_threshold`	Y	(minutes) Exclude rules from the benchmarking report if elapsed time is below this threshold
`somalier`	Y	Check relatedness and sex discordance with Somalier (requires sex_linker.tsv) only available in full run mode. Support of Mac OSX is experimental, so you may want to set this to False on a Mac

The remainder of the `config.yaml` file contains a selected set of exposed per-tool parameters. For the most part, this allows tuning of resource allocation on a per-tool basis (i.e. `threads` and `memory` in MB). Java-based tools also allow for arbitrary java options to be passed through via `java_opts`. Additional exposed parameters include:

- `genomicsDBImport` and `genotypeGVCFs`: We have exposed some useful parameters that have been helpful to adjust as scale increases. Please see GATK documentation for the relevant tools to learn more.
- `verifyBamID`: A `region` field allows the user to specify chromosomes over which to run contamination analysis, in an attempt to mitigate large memory requirements.

1.4 Execution

1.4.1 Deploying the pipeline

With the `config.yaml` configured to your run-mode of choice with paths to the necessary manifest and input files, the workflow can be executed on any infrastructure using the `snakemake` command, supplied with further Snakemake command-line arguments (e.g. specifying a profile with `--profile` or `--cluster` to submit jobs to an HPC) depending on your environment.

Test your configuration by performing a dry-run:

```
snakemake --use-conda -n
```

Execute the workflow locally via:

```
snakemake --use-conda --cores $N
```

Execute the workflow on a cluster using something like:

```
snakemake --use-conda --cluster sbatch --jobs 100
```

The pipeline will automatically create a subdirectory for logs in `logs/` and temporary workspace at the path specified for `tempDir` in the `config.yaml`.

1.4.2 Wrapper scripts

We have provided two convenience scripts in the 54gene-wgs-germline repository to execute the workflow in a cluster environment: `run.sh` and `wrapper.sh`. You may customize these scripts for your needs, or run using a profile (e.g. this profile for a slurm job scheduler).

The `wrapper.sh` script embeds the `snakemake` command and other command-line flags to control submission of jobs to an HPC using the `cluster_mode` string pulled from the `config.yaml`. This script also directs all stdout from Snakemake to a log file in the parent directory named `WGS_${DATE}.out` which will include the latest git tag and version of the pipeline, if cloned from our repository. For additional logging information, see *Logging*.

This wrapper script can be edited to your needs and run using `bash run.sh`.

1.4.3 Automatic retries with scaling resources

Many rules in this pipeline are configured to automatically re-submit upon failure up to a user-specified number of times. This is controlled via Snakemake's `--restart-times` command line parameter. The relevant rules will automatically scale resource requests with every retry as follows (example from `rule align_reads`):

```
resources:
   mem_mb=lambda wildcards, attempt: attempt * config["bwa"]["memory"],
```

In this example, if the specified amount for `bwa` used in `align_reads` is set to `memory: 3000` but the job fails, it will be resubmitted on a second attempt with twice the memory. Subsequently, it if fails again, a third attempt with three times the memory will be submitted (depending on your setting for `--restart-times`). If your system or infrastructure does not have the necessary memory available, there is potential for re-submission of jobs to fail due to insufficient resources.

1.4.4 Logging

All job-specific logs will be directed to a `logs/` subdirectory in the home analysis directory of the pipeline. This directory is automatically created for you upon execution of the pipeline. For example, if you run the pipeline on a Slurm cluster with default parameters, these log files will follow the naming structure of `snakejob.<name_of_rule>.<job_number>`.

If you choose to use the `wrapper.sh` script provided and modified for your environment, a `WGS_${DATE}.out` log file containing all stdout from snakemake will also be available in the parent directory of the pipeline.

1.5 Investigate the results

1.5.1 Assessing completion

Upon pipeline completion, verify that all steps have completed without error by checking the top-level log (called `WGS_<datestamp>.out` if using the optional wrapper script; otherwise see Snakemake's documentation for the default location of stdout). The bottom few lines of the file should contain something similar to `nnn of nnn steps (100%) done`. Additional job logs (when run on a high-performance computing cluster) are stored in the `logs/` sub-directory.

1.5.2 Outputs and results

All pipeline results are stored in the `results/` directory.

- The hard-filtered, joint-called VCF can be found in `results/HaplotypeCaller/filtered/HC_variants.hardfiltered.vcf.gz`
- For future joint-calling, the gVCFs are located at `results/HaplotypeCaller/called/<sample>_all_chroms.g.vcf.gz`
- Deduplicated and post-BQSR bams are found at `results/bqsr/<sample>.bam`

Samples that fail the following thresholds are automatically removed from the above joint-called VCF, and the output is placed in `results/post_qc_exclusions/samples_excluded.HC_variants.hardfiltered.vcf.gz`. The record of sample exclusions, along with reasons for exclusion, is found at `results/post_qc_exclusions/exclude_list_with_annotation.tsv`. Values listed are defaults, but can be changed in the `config.yaml`.

1. Average depth of coverage < 20x
2. Contamination > 3%
3. Het/Hom ratio > 2.5

1.5.3 QC

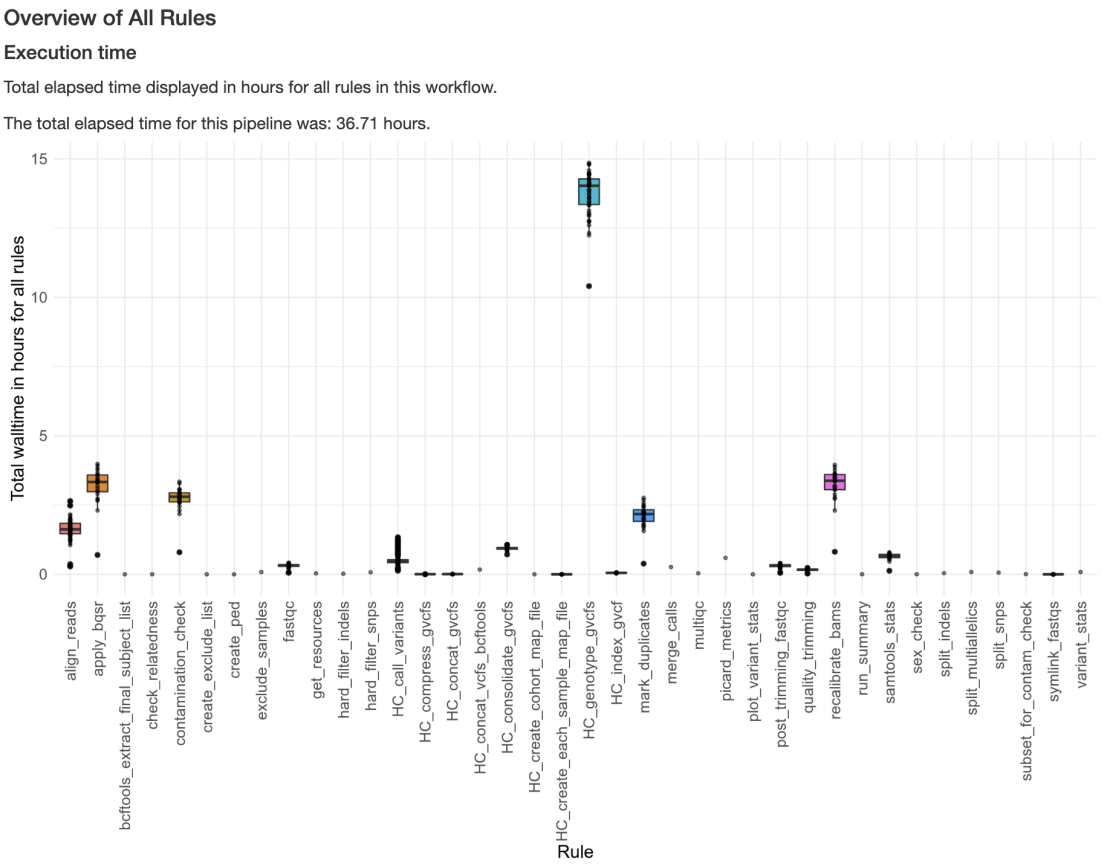
The following QC metrics are available (depending on run mode selected):

- Pre- and post-trimming FastQC reports at `results/fastqc/` and `results/post_trimming_fastqc/`, respectively
- Trimming stats via fastp at `results/paired_trimmed_reads/`
- Alignment stats via samtools at `results/alignment_stats/`
- Recalibration stats from bqsr at `results/bqsr/`
- Relatedness via Somalier at `results/qc/relatedness/`

- Sample contamination via verifyBamID at `results/qc/contamination_check/` (for full runs only; **not** included in joint-genotyping only run mode)
- Inferred sex via bcftools +guess-ploidy at `results/qc/sex_check/`
- Picard metrics at `results/HaplotypeCaller/filtered/`
- bcftools stats at `results/qc/bcftools_stats/`
- MultiQC report at `results/multiqc/`
- Benchmarking report of pipeline performance statistics (i.e. elapsed time, memory and CPU utilization for rules above specified `time_threshold` in `config.yaml`) at `performance_benchmarks/benchmarking_report.html`
- Run summary report for the pipeline, excluded samples and discordances at `results/run_summary/run_summary.html`

Examples

Below is an example of a plot from the `benchmarking_report.html` report generated, showing execution time across rules in a pipeline run:



Below is an example of the final subject tracking table generated in the `run_summary.html` report, showing QC outcomes for subjects included in a pipeline run:

Final Subject Tracking

A total of 24 subjects were queued up for analysis in this run; 23 subjects passed automated QC and are present in the final VCF. The table below reports lane/read failures for select FastQC metrics. Only failing fastqs are reported in those columns. Please see the MultiQC report for additional information.

Summary of Subject Fate

Subject	QC Outcome for This Run	Per Base Sequence Quality Failures	Per Base N Content Failures	Overrepresented Sequences Failures	Rerun Recommendation	Coverage
Sample01	Pass				Pass	24.8
Sample02	Pass				Pass	29.3
Sample03	Pass				Pass	24.6
Sample04	Pass				Pass	32.9
Sample05	Pass				Pass	30.2

1.6 Tests

1.6.1 Unit tests

Unit tests for the python modules used in this workflow can be found in `workflow/scripts/tests` and run using Pytest which is included in the conda run-time environment for this pipeline.

To run all available unit tests:

```
conda activate 54gene-wgs-germline
pytest -s workflow/scripts/tests/*.py
```

1.6.2 Pipeline/Integration tests

To test the core pipeline, we provide a small test dataset and instructions on how to use this dataset available in a repository here.

To summarize, this test dataset contains a small region of chromosome 21 from the NA12878 platinum reference genome. The above repository contains all necessary inputs (configuration file, manifest, intervals, sex_linker files) required to run the pipeline in all three run-modes. The README provides instructions on how to use these files to execute a test using the 54gene-wgs-germline pipeline.

1.6.3 Snakemake unit tests

In development (TBD)

1.6.4 CI/CD

The aforementioned python unit tests and integration tests (in all three run-modes) are run as part of the Gitlab Continuous Integration (CI) pipeline for this codebase. You can find the status of the CI pipeline on the main repository page.

Note: The test suite and CI pipeline are still a work in progress.

1.7 Core pipeline

This page describe details of the various run-modes available in this pipeline, the rules used within them and further specifications on the tools used. This page provides information on the parameters used for certain tools and behaviours between the run-modes.

1.7.1 Pulling in resources

There is no config option to point to a reference genome. The pipeline automatically pulls in the required GRCh38 reference files, including the reference genome and all requisite indices as well as the known SNP and indel files for running BQSR, from the Broad Institute's public AWS S3 bucket (`s3://broad-references/hg38/v0/`) in the `rule get_resources`. We have not provided an option for hg19/GRCh37.

1.7.2 FastQC and read trimming

In full and fastqc_only run modes:

The pipeline will create symbolic links for the supplied input FASTQs from the manifest file in `rule symlink_fastqs`. FastQC generates reports based on filename, and has no option to change the output filenames. Symlinking allows harmonization of filenames to the convention followed throughout the pipeline; for FASTQs, that convention is `<readgroup>_r[12].fastq.gz`. Please bear in mind these symlinks when managing data on your infrastructure.

`rule fastqc` will then generate FastQC reports for all input FASTQs. Note that this is one of the rules governed by the `max_concurrent` config argument (see *Config parameters*). On filesystems where IO bandwidth is capped, you may want to control the number of concurrent rules running at this stage.

Next, we perform read trimming and adapter removal (currently hard-coded to use Illumina's TruSeq adapters) using fastp. If you need to use alternate adapters or adjust other `fastp` parameters, please submit a feature request to expose these as parameters in config space.

Post-trimming, FastQC will be run again on the trimmed reads. This results in FastQC results for the raw input reads in `results/fastqc`, and post-trimmed reads in `results/post_trimming_fastqc`. Review these reports to monitor read quality and effective adapter removal.

Note that **fastqc_only** run mode will stop here, allowing a quick turnaround in sharing read quality information with lab, and assessing whether there are any samples to drop before performing a more computationally costly full run.

1.7.3 Read alignment, deduplication, and BQSR

In full mode only:

Trimmed reads are aligned to the reference genome using BWA in `rule align_reads`. The read1 and read2 data are combined into a single output BAM per FASTQ pair. If samples were run over several lanes (e.g. 4 lanes), each per-lane read1 and read2 FASTQ pair will be aligned individually, then combined during the subsequent deduplication step (see `rule mark_duplicates`). This helps with efficient alignment by running multiple smaller alignments in parallel. The read group IDs of the BAM files will include the sequencing center ID specified in the `config.yaml` under the `center_id` parameter.

The alignment step outputs aligned and sorted BAMs, one per sample-and-lane, at `results/mapped/<readgroup>.bam`. These BAMs are flagged as temp, so they are automatically removed unless run with the `--notemp` Snakemake flag (see Snakemake documentation).

After alignment, duplicate reads in the sorted BAMs generated for each readgroup are then marked and removed in `rule mark_duplicates`. It is at this step that samples split over multiple lanes will be merged, and subsequently named with the sample ID provided in the manifest. This generates one BAM file for each sample, found as `results/dedup/<sample ID>.bam`. Subsequently, the pipeline will use GATK's BaseRecalibrator to generate a recalibration table for base quality scores using known sites VCF pulled from the Broad's resources bucket. This recalibration is then applied to each BAM in `rule apply_bqsr`. Samtools stats are then generated for all BAMs. See the *Investigate the results* page for more information on where to find the stats. Upon recalibration, the per-sample, sorted BAM files and their indexes can be found in `results/bqsr/<sample ID>.bam`.

1.7.4 Variant calling

Per-sample gVCFs are generated in `rule HC_call_variants` using GATK's HaplotypeCaller. Calling is parallelized over a user-specified number of intervals and/or regions of the genome using the interval file listed in the config. A temp-flagged gVCF for each sample will be generated for each specified interval/region; these are automatically cleaned up once they are all present and have been successfully combined into a single per-sample gVCF using GATK's GatherVcfs. This method allows for parallelization and reduction in overall execution time for variant calling. Following the GVCF workflow, these are to be used for joint genotyping of multiple samples later in the pipeline for scalable analyses. The resulting per-sample gVCF is compressed and indexed, and can be found at `results/HaplotypeCaller/called/<sample>_all_regions.g.vcf.gz`.

1.7.5 Joint genotyping

In joint_genotyping mode only:

It is at this step in the workflow that a second entry point is provided when the run mode in the `config.yaml` is set to `joint_genotyping`. In this run mode, the gVCFs provided in the manifest file and their indices will be symlinked to a subdirectory within `/results/HaplotypeCaller/called/` prior to continuing on to the rest of the workflow.

In joint_genotyping and full run modes:

In order to perform joint-genotyping over multiple samples using GATK's GenotypeGVCFs, the input gVCFs must be consolidated across samples as the genotyping step can only take one single input. To circumvent this issue, we use GATK's GenomicsDBImport in `rule HC_consolidate_gvcfs` to generate database stores for each sample, parallelized again across intervals/regions, to then pass into GenotypeGVCFs. DBImport can potentially take up a lot of temp space so it is recommended that `--tmp-dir` be used to redirect to a larger temp space. The `--batch-size` and `--reader-threads` parameters can be tweaked in the `config.yaml` to read in more data stores concurrently or in larger batch sizes but the default settings are those suggested by GATK developers.

Joint genotyping using the various database stores created is performed in `rule HC_genotype_gvcfs` to emit a genotyped gVCF for each interval/region in `results/HaplotypeCaller/genotyped/{interval}.vcf.gz`. The `--max_alt_alleles` to genotype and `--max_genotype_count` for each site can be tweaked in the `config.yaml`.

We exposed these and other parameters for GenomicsDBImport after encountering recent issues where the `--max-alternate-alleles` flag for GenotypeGVCFs was set at a default of 6 but was not actually being applied as a threshold. A fix in GATK v4.2.4.1 attempted to apply this threshold, but instead resulted in a bug where the tool would crash upon reaching a locus exceeding this threshold. Subsequently, an update in v4.2.5.0 introduced a new parameter for GenotypeGVCFs called `--genomicsdb-max-alternate-alleles`, which is required to be minimum one greater than `--max-alternate-alleles` to account for the NON_REF allele.

The per-interval/region, genotyped gVCFs will be concatenated into one sorted, indexed, project-level multi-sample gVCF for downstream analysis in `results/HaplotypeCaller/genotyped/HC_variants.vcf.gz`.

Note: While GenomicsDBImport supports adding N+1 samples to the datastores, our pipeline does not utilize this functionality and instead creates the databases every time from scratch. This was a development choice made to avoid issues with potential failures with maintaining the datastores and revisiting them in future analyses.

1.7.6 Variant filtering

The project-level VCF is normalized and multiallelics are split using `bcftools norm` in `rule split_multiallelics`. This means that the resulting VCF may have multiple lines representing the same genomic position. This is conformant with VCF specifications, and may not be expected as input by all downstream tools. We have elected to split multiallelics for several reasons, including:

- Inability to apply hard filtering to multi-type loci. GATK's hard filters require first splitting indels and SNPs; multi-type loci don't get split into either category. So, by splitting multiallelics, you can apply the appropriate filter to all alt alleles
- Difficulty in parsing which annotations refer to which allele after using a tool like VEP or SNPeff

Hard-filtering using GATK's VariantFiltration tool is performed separately on the SNP and indel-specific project-level VCFs in `rule hard_filter_snps` and `rule_hard_filter_indels`. After variants are flagged in the FILTER column based on hard filters, indels and snps are recombined and can be found at `results/HaplotypeCaller/filtered/HC_variants.hardfiltered.vcf.gz`. For more information on how we perform hard-filtering, see GATK's documentation on hard-filtering recommendations.

Note: We currently do not remove the filtered sites themselves from the VCF but instead just update the filter field. You will want to do a pass with GATK or bcftools to filter out non-PASS variants.

1.7.7 Post-calling QC

Contamination Check

In full mode only:

As an added QC measure, we perform a contamination check on the BAM files using a tool called VerifyBamID. This tool estimates the most likely proportion of contaminant DNA present in a sample given phred likelihoods of actual basecalls, assuming HWE.

The tool normally takes the entire BAM file as an input but to reduce the computational burden of performing this check, we opted to only subset particular chromosomes (ideally one or two) from the BAM files to perform the check. We have found that is this sufficient for initial flagging of contamination for further in-depth investigation of troublesome samples. We allow the ability to select these chromosomes within the `config.yaml`.

This step in `rule contamination_check` will output various contamination metrics for each sample BAM file that are combined in a summary file. This summary file will be later used for automated filtering of samples out of the project-level VCF based on thresholds defined in the `config.yaml`. See the *Sample exclusions* section for more information.

Checking relatedness with Somalier

If `check_relatedness` is set to `yes` in the `config.yaml`, the pipeline will run Somalier to check for relatedness amongst the samples. Somalier is a tool that can be used to check any number of samples from joint-called VCFs for identity and to infer relationships. The tool takes as input a jointly-called cohort VCF and PED file of expected sexes and relationships. Our pipeline requires a simple sex linker file described in *Configuration* and creates the PED file for you. An example of the Somalier output can be found here.

This tool provides a rough estimate of relatedness which we mainly use to identify unexpected genetic duplicates. To confirm specific relationships, we perform a second pass evaluation of the relevant samples using more specialized software, e.g. KING, graf, etc. Somalier uses the following equation to determine relatedness:

```
(shared-hets)(i,j)-2*ibs0(i,j)/min (hets(i),hets(j))
```

This assumes, as noted in their publication, that the sites they've selected on which to assess relatedness are "high-quality, unlinked sites with a population allele frequency of around 0.5." We suspect this will not hold true across all populations, and we are currently working in a relatively underrepresented ancestry group. It is unclear how much this will degrade across multiple populations with some degree of shared ancestry. Note that the relatedness value will always be depressed when comparing samples from disparate ancestries, for example, NA12878 with continental African subjects.

Sex Check

Somalier also provides functionality to assess sex discordance. The HTML report provided by Somalier, and in the MultiQC report that ingests this data, includes a plot of scaled mean depth on X vs. self-reported sex. This plot allows quick identification of disagreement between reported and genetic sex.

In addition to Somalier, we also use bcftools' guess-ploidy plugin to determine sample sex from genotype likelihoods. These results are also included in the MultiQC report generated at the end of the post-calling QC stage. See *MultiQC* for more information.

Sample exclusions

We exclude samples from the project-level hard-filtered VCF in `rule create_exclude_list` based on metrics and information generated from the contamination check and bcftools stats. Samples are excluded based on the following default thresholds:

- Max het/hom ratio of 2.5
- Minimum average depth of 20
- Maximum contamination estimate of 0.03 (only used if run in full run mode)

These thresholds can be tweaked in the `config.yaml`. A list of samples to exclude and another list with these samples and annotations for why they were excluded will be generated in `results/post_qc_exclusions/`.

Post sample exclusion, another sorted and indexed, project-level, hard-filtered VCF will emitted in `results/post_qc_exclusions/samples_excluded.HC_variants.hardfiltered.vcf.gz`. Note that the ID column here will also be updated to `CHROM:POS:REF:ALT` using bcftools annotate.

MultiQC

A MultiQC report is generated for all three run-modes and will differ in content depending on which post-calling QC checks were performed.

For **fastqc_only** run mode, the multiQC report will include:

- Pre- and post-read-trimming fastQC results

For the **full** run mode, the multiQC report will include:

- Pre- and post-read-trimming fastQC results
- Bcftool stats on joint-called variants
- Deduplication metrics for BAM files
- Sex check results from bcftools guess-ploidy
- Contamination check results from verifyBamID
- If specified in config, relatedness check results from Somalier
- Variant calling metrics

For **joint_genotyping** mode, the multiQC report will include:

- Variant calling metrics
- Sex check results from bcftools guess-ploidy
- Bcftool stats on joint-called variants
- If specified in config, relatedness check results from Somalier

1.8 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.8.1 [1.0.0] - 2022-08-01

Added

- Initial release of basic feature set, including the following
- Read filtering and trimming
- Read alignment, deduplication, and BQSR
- Variant calling and filtering
- Joint-genotyping
- Sex discordance and relatedness assessment
- Generate MultiQC reports
- Multiple run modes for initial QC, full end-to-end runs, and joint-calling only
- Read the Docs documentation
- Companion test data repo

1.9 For developers

1.9.1 Contribute back

For developers, if you would like to contribute back to this repository, consider forking the original repo and creating a pull request:

1. Fork the original repo to a personal or lab account.
2. Clone the fork to your local system, to a different place than where you ran your analysis.
3. Copy the modified files from your analysis to the clone of your fork, e.g., `cp -r workflow path/to/fork`. (Make sure to not accidentally copy config file contents or sample sheets. Instead, manually update the example config files if necessary)
4. Commit and push your changes to your fork.
5. Create a pull request against the original repository.

1.9.2 Obtain updates from upstream

Whenever you want to synchronize your workflow copy with new developments from upstream, do the following.

1. Once, register the upstream repository in your local copy: `git remote add -f upstream git@github.com:snakemake-workflows/54gene-wgs-germline.git` or `git remote add -f upstream https://github.com/snakemake-workflows/54gene-wgs-germline.git` if you do not have setup ssh keys.
2. Update the upstream version: `git fetch upstream`
3. Create a diff with the current version: `git diff HEAD upstream/master workflow > upstream-changes.diff`
4. Investigate the changes: `vim upstream-changes.diff`
5. Apply the modified diff via: `git apply upstream-changes.diff`
6. Carefully check whether you need to update the config files: `git diff HEAD upstream/master config`. If so, do it manually, and only where necessary, since you would otherwise likely overwrite your settings and samples.

1.10 References

The following software packages are used in this pipeline:

Software	Website	Citation
AWS CLI	https://github.com/aws/aws-cli	
BCFtools	https://github.com/samtools/bcftools	doi: 10.1093/gigascience/giab008
BWA	https://github.com/lh3/bwa	doi: 10.48550/arXiv.1303.3997
conda	https://docs.conda.io/en/latest/	
fastp	https://github.com/OpenGene/fastp	doi: 10.1093/bioinformatics/bty560
FastQC	https://www.bioinformatics.babraham.ac.uk/projects/fastqc/	
GATK4	https://github.com/broadinstitute/gatk	
matplotlib	https://matplotlib.org/	
MultiQC	https://github.com/ewels/MultiQC	doi: 10.1093/bioinformatics/btw354
pandas	https://pandas.pydata.org/	
Python	https://www.python.org/	
R	https://www.r-project.org/	R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, (2022).
SAMtools	https://github.com/samtools/samtools	doi: 10.1093/gigascience/giab008
Snakemake	https://github.com/snakemake/snakemake	doi: 10.12688/f1000research.29032.1
Somalier	https://github.com/brentp/somalier	doi: 10.1186/s13073-020-00761-2
Tabix	https://github.com/samtools/htslib	doi: 10.1093/gigascience/giab007
VerifyBamID	https://github.com/statgen/verifyBamI	doi: 10.1016/j.ajhg.2012.09.004