User Manual for ProteomeGenerator 2

November 8, 2022

https://github.com/kentsisresearchgroup/ProteomeGenerator2

kentsisresearchgroup@gmail.com

Introduction

ProteomeGenerator 2 (PG2) is a versatile, fully Linux-compatible, cluster-oriented pipeline for integrated proteogenomic analysis built on the Snakemake infrastructure. It expands the first version of ProteomeGenerator https://github.com/jtpoirier/proteomegenerator to incorporate sample-specific DNA sequence analysis, either from fastq, bam, or VCF files, and merge it with transcriptome analysis using sample-specific RNA-sequencing data to generate sample-specific proteomes for subsequent integrative mass spectrometry (MS) proteomics analysis.

PG2 is compatible with diverse sequence and mass spectral analysis algorithms. MaxQuant has been bundled into the pipeline (chosen for its open-source integration in clustered Linux environment, as enabled by UltraQuant https://github.com/kentsisresearchgroup/UltraQuant and other implementations). The individualized databases generated by PG2 can be used with any MS searching algorithm.

Overview of ProteomeGenerator 2

Briefly, proteome generation works as follows: first, variant calls are used to "patch" the reference genome, yielding one that is specific to the sample(s) in question. Next, RNA-seq reads are aligned (STAR) to the newly individualized genome, from which the complete transcriptome is assembled (StringTie) in either a *de novo* or annotation-guided manner and corresponding nucleotide sequences are read out. Finally, the nucleotide sequences are 6-frame translated, the most probable open reading frame(s) for each transcript are selected based on length of longest contiguous peptide sequence and homology to known proteoforms, and mapped back to the genome to produce the individualized proteome database. By combining arbitrary sets of sequencing/structural/etc variant calls with *de novo* transcriptome assembly, PG2's databases are able to capture a diverse and extensible breadth of genomic and transcriptomic variation.

In addition to its primary proteome generation functionality, PG2 also packages in *UltraQuant*, a *Linux*-compatible plug-n-play installation of the popular proteomics suite *MaxQuant* (achieved by leveraging a *Singularity* container running a compatible version of *Mono)*. Furthermore, post-proteomics modules are provided to map identified non-canonical / unannotated peptides back to the genetic events from which they derive. Finally, implementations of the *GATK Best Practices* pipelines for WGS/WES alignment and germline/somatic short variant calling are also bundled into the release, allowing the entire proteogenomic analysis, from raw sequencing data to novel protein identifications and mapping, to be carried out in a single command.

Calculation Modes / Workflows

Genome personalization

A key feature of PG2 is the ability to construct a personalized (customized) genome, specific to the individual of study, in order to subsequently generate the precise protein sequences that comprise that individual's proteome. **To invoke this workflow, WGS/WES (DNA) sequencing data must be provided as input.**

Given this input data, PG2 first aligns this data (BWA) to a provided reference genome (e.g., GRCh38 FASTA+GTF) and then utilizes the GATK variant callers HaplotypeCaller and Mutect2 in order to genotype the individual's genome and identify somatic variants[1]. This sample-specific information is then used to 'patch' the reference genome to yield the personalized genome (FASTA+GTF). This new genome consists of two parallel 'haplotype' sequences, to fully capture the diploid nature of the genotyped genome.

"De novo" transcriptome assembly

PG2 incorporates "de-novo" transcriptome assembly in order to capture novel or non-canonical sequences/transcripts/isoforms that are not included in the reference genome annotation. **To invoke this workflow, RNA-seq data must be provided as input.**

Given this input data, PG2 first aligns (via STAR) the RNA-seq reads to the genome (either the personalized or reference genome, depending on whether genome personalization was invoked). Next, the aligned reads are assembled (via StringTie) into a complete and precise set of non-redundant transcripts comprising the sample-specific transcriptome (from which the corresponding nucleotide sequences are extracted in the next step).

If using a custom genome, then the extracted transcript sequences will contain the precise nucleotide sequences (including sample-specific mutations) present in the individual. If using a reference genome, the extracted sequences will contain reference nucleotide sequences. Importantly, however— in either case, non-canonical transcripts/isoforms (e.g., novel splicing, skipped exons, *de novo transcripts,* etc) WILL be represented. The difference is whether sample-specific mutations are contained in the extracted sequences or not.

Protein sequence (proteome) generation

This segment begins with extraction of transcript (nucleotide) sequences. If transcriptome assembly was invoked (i.e., RNA-seq data was provided as input) as a prior step, then transcript sequences are extracted from the newly assembled sample-specific transcriptome. If transcriptome assembly was NOT

[1] If a *matched normal sample* is provided in addition to the tumor/experimental sample, somatic variants are identified by comparing the two samples. If only the tumor sample is provided, somatic variants are identified primarily by allele frequency analysis (see GATK Mutect2 spec for full details).

invoked as a prior step, then transcript sequences are extracted directly from the custom genome annotation.

The transcripts are then 6-frame translated into amino acid sequences, and the most ‘promising’ sequences (based on length, homology, or other customizable parameters) are retained in the output proteome.

Modules

PG2 is organized into the following modules, each of which can be used on their own, or invoked (recursively) as a subworkflow/submodule:

- ProteomeGenerator2.py (RNA-seq / main proteome generation module) : the main executable file of PG2. This module contains the rules for RNA-seq mapping, transcriptome and fusion transcript assembly, protein prediction, and MaxQuant. It can optionally invoke the genome individualization module as a *subworkflow* (meaning it runs **before** the module’s own rules run). It also imports the novel analysis module rules for post-proteomics search analysis.

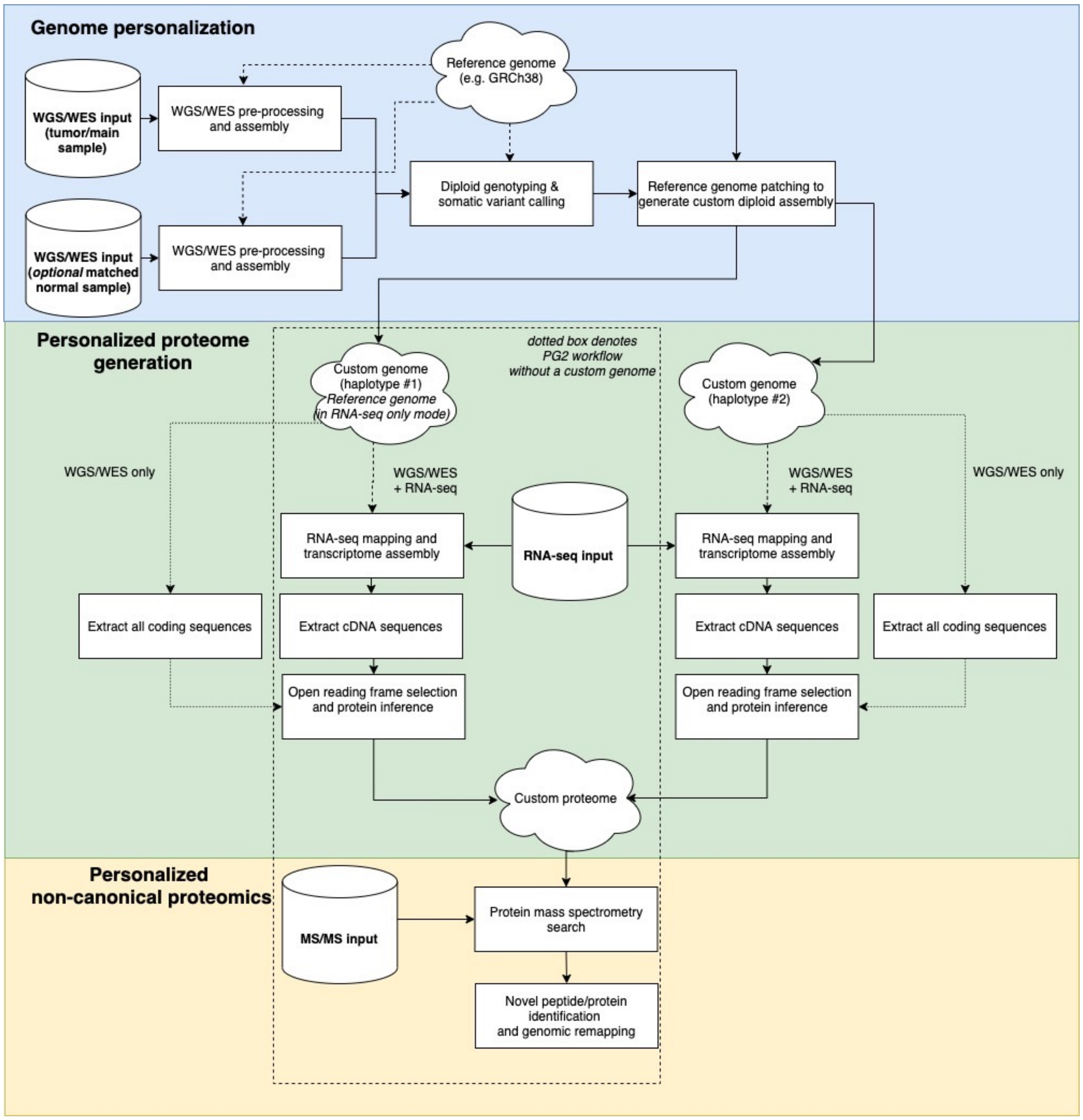
- genome_personalization_module.py : contains the rules for taking a VCF file and creating an individualized bi-haploid genome (sequence FASTA + annotation GTF) for downstream analysis. It can optionally invoke the variant calling module as a subworkflow, or ingest a premade VCF.

- WGS_variant_calling.py : contains the rules for taking a pre-processed, analysis-ready BAM file and calling variants to produce a germline and/or somatic VCF that can be used for genome individualization (or other purposes). It can optionally invoke the WGS pre-processing module as a subworkflow, or ingest a pre-processed BAM.

- WGS_preprocessing.py : contains the rules for aligning and pre-processing raw FASTQ reads or an unprocessed BAM file in preparation for variant calling. Will typically be invoked by the variant calling module as a subworkflow.

- novel_analysis.py : contains the rules for aggregating novel peptides identified on MaxQuant proteomics search (as well as mapping them back to their originating mutations). Also in development are rules for deconvoluting non-mutational novel peptides (from de novo isoforms, upstream start sites, upstream ORFs, etc) and for rudimentary protein expression analysis.

Graphical scheme of PG2


Genome personalization
WGS/WES input (tumor/main sample)
WGS/WES pre-processing and assembly
Reference genome (e.g. GRCh38)
Diploid genotyping & somatic variant calling
Reference genome patching to generate custom diploid assembly
WGS/WES input (optional matched normal sample)
WGS/WES pre-processing and assembly
Personalized proteome generation
dotted box denotes PG2 workflow without a custom genome
Custom genome (haplotype #1) Reference genome (in RNA-seq only mode)
Custom genome (haplotype #2)
WGS/WES only
WGS/WES + RNA-seq
RNA-seq input
WGS/WES + RNA-seq
WGS/WES only
RNA-seq mapping and transcriptome assembly
RNA-seq mapping and transcriptome assembly
Extract all coding sequences
Extract cDNA sequences
Extract cDNA sequences
Extract all coding sequences
Open reading frame selection and protein inference
Open reading frame selection and protein inference
Custom proteome
Personalized non-canonical proteomics
MS/MS input
Protein mass spectrometry search
Novel peptide/protein identification and genomic remapping

Installation

Required packages

- *Miniconda for Python 3* (version >= 4.8.3)
- *Snakemake* (version >= 5.4.5)
- *Singularity* (version >= 3.6.0)
- *MaxQuant* (version = 1.6.2.3)

All other packages and libraries will be managed by *Snakemake* (via conda) and do not need to be installed by the user.

Recommended Installation Steps

1. Install *Singularity*
    a. This is already installed on the Lilac cluster, and most likely on most Linux clusters. To verify this, enter 'module avail' and confirm that *Singularity* is on this list. You can then load *Singularity* with 'module load singularity/<version>'
2. Install *Miniconda for Python 3* -- https://conda.io/en/latest/miniconda.html
    a. Copy the Linux installer link to clipboard
    b. In Lilac terminal, enter 'wget <installer_link>' to download the installer shell script
    c. Enter './<installer_sh>' to run the installation script, make sure to enter yes when asked if conda should be added to the PATH
3. Install mamba (conda repository manager)
    a. `conda install -c conda-forge mamba`
4. Install snakemake as a new conda environment
    a. `amba create -c conda-forge -c bioconda -n snakemake snakemake=5.4.5`
5. Activate the snakemake environment
    a. conda activate snakemake
6. Download PG2
    a. git clone https://github.com/kentsisresearchgroup/ProteomeGenerator2.git
7. Download MaxQuant into PG2 directory
    a. cd ProteomeGenerator2/
    b. git clone https://github.com/kentsisresearchgroup/UltraQuant.git

Input Data

The fully incorporated pipeline utilizes the following data:

1. Paired-end whole-genome/whole-exome sequencing data OR variant call data
   (Used to create the individualized genome that produces individualized protein sequences)
   a. WGS/WES data can be in either FASTQ/BAM format, or both
   b. Variant calls should be in VCF format *(untested as of now)*

2. RNA-seq data (paired- or single-end)
   a. FASTQ/BAM format, or both

3. Raw MS/MS data
   a. ThermoFisher RAW files
   b. Other formats (e.g., mzML, MGF) should also work (albeit untested)

Based on the workflow defined by the user in the configuration file, not all types of input data will necessarily be used in a given run.

Configuration File (configfile)

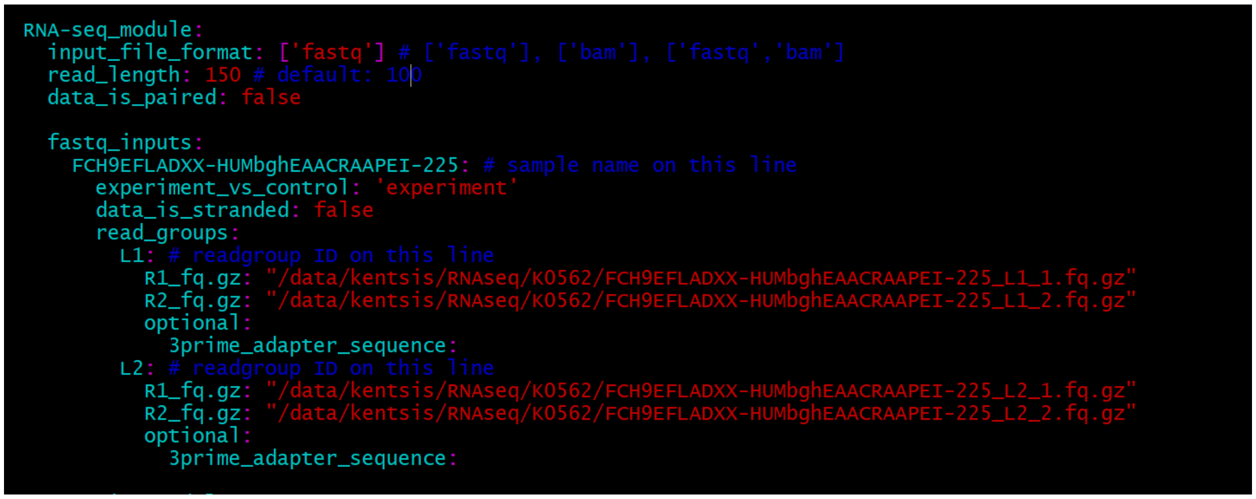
The configfile is a YAML file that is written in an intentionally verbose style to make it easy for non-technical users. It is a hierarchically organized file (with parent-child relationships denoted by a two-space indentation [SPACEBAR twice]) whose top-level categories are as follows:

- **Directories**
- **"Stock" genome and proteome references**
- **User-defined workflow**
- **Input files**
- **Parameters**

Additionally, the documentation/comments provided in the configfile itself is verbose, so please consult it directly. We provide a few templates and examples in the configfiles/ directory. Please make sure to read through the entire configfile the first time through, to get a full idea of the parameters that are configurable by the user.

In general, the structure of the configfile should not be changed. Doing so will usually cause the program to break. The exception to this is in the input files section, for which sample/readgroup/file/etc subsections should be duplicated to accommodate multiple entries. A couple of examples:

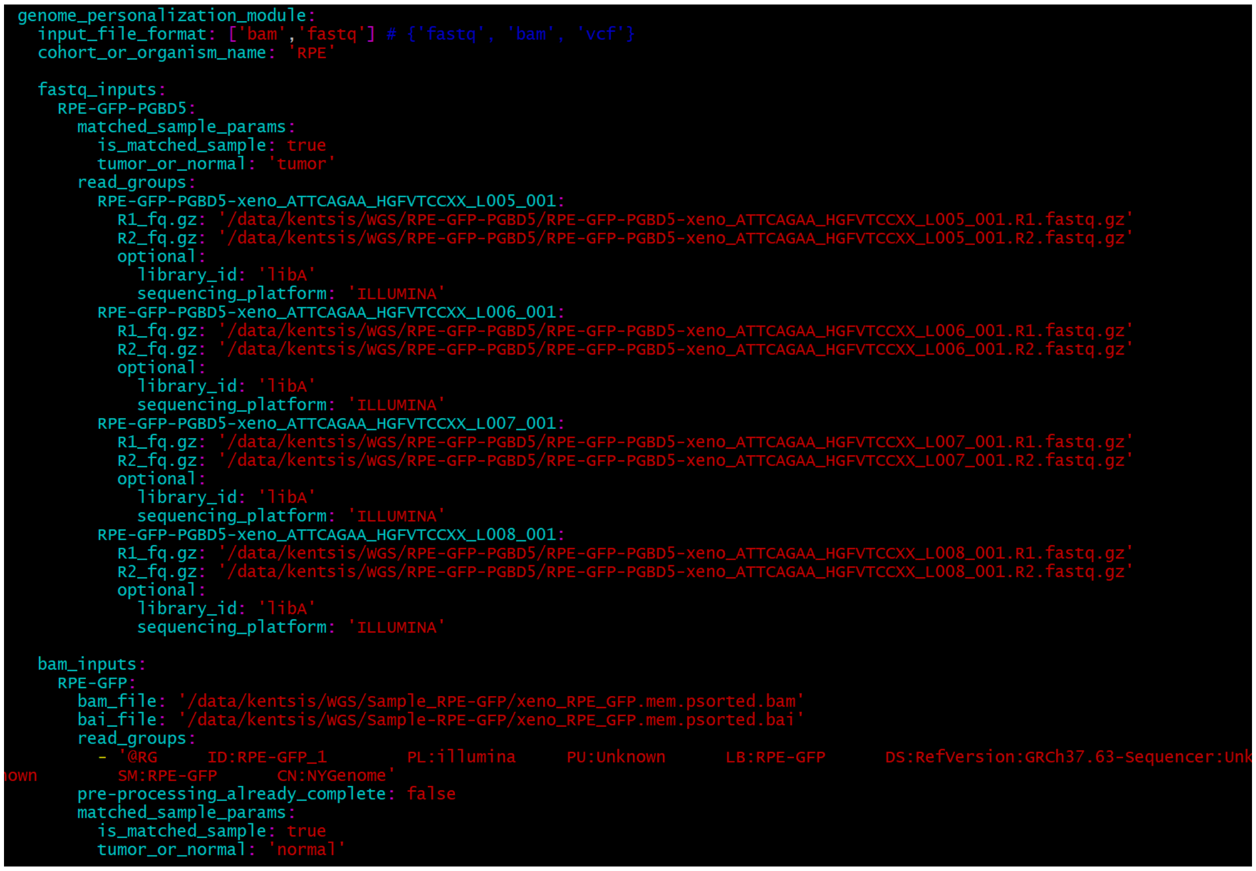
Multiple read groups (L1, L2)



```
RNA-seq_module:
  input_file_format: ['fastq'] # ['fastq'], ['bam'], ['fastq','bam']
  read_length: 150 # default: 100
  data_is_paired: false

  fastq_inputs:
    FCH9EFLADXX-HUMbghEAACRAAPEI-225: # sample name on this line
      experiment_vs_control: 'experiment'
      data_is_stranded: false
      read_groups:
        L1: # readgroup ID on this line
          R1_fq.gz: "/data/kentsis/RNAseq/K0562/FCH9EFLADXX-HUMbghEAACRAAPEI-225_L1_1.fq.gz"
          R2_fq.gz: "/data/kentsis/RNAseq/K0562/FCH9EFLADXX-HUMbghEAACRAAPEI-225_L1_2.fq.gz"
          optional:
            3prime_adapter_sequence:
        L2: # readgroup ID on this line
          R1_fq.gz: "/data/kentsis/RNAseq/K0562/FCH9EFLADXX-HUMbghEAACRAAPEI-225_L2_1.fq.gz"
          R2_fq.gz: "/data/kentsis/RNAseq/K0562/FCH9EFLADXX-HUMbghEAACRAAPEI-225_L2_2.fq.gz"
          optional:
            3prime_adapter_sequence:
```

Multiple input file types (FASTQ, BAM) + multiple read groups

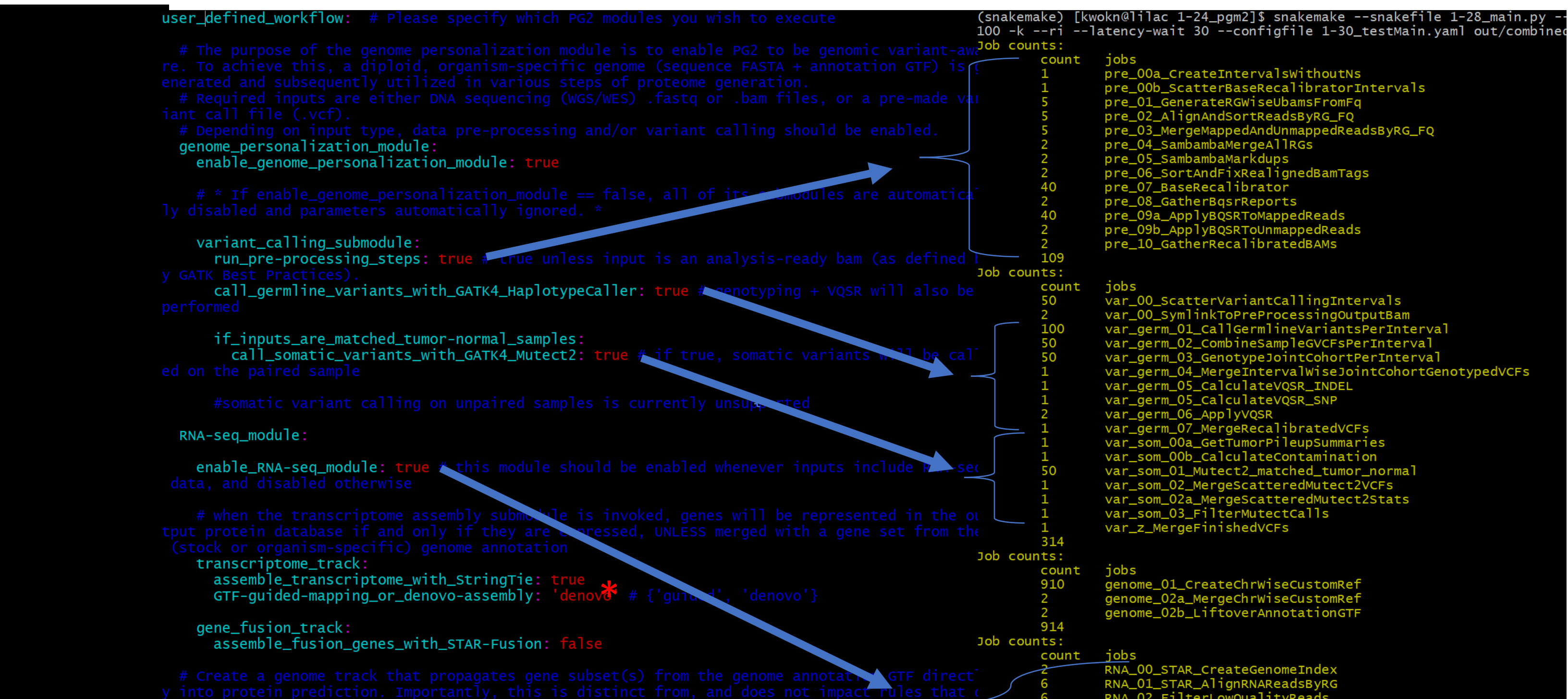


```
 genome_personalization_module:
   input_file_format: ['bam','fastq'] # {'fastq', 'bam', 'vcf'}
   cohort_or_organism_name: 'RPE'

   fastq_inputs:
     RPE-GFP-PGBD5:
       matched_sample_params:
         is_matched_sample: true
         tumor_or_normal: 'tumor'
       read_groups:
         RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L005_001:
           R1_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L005_001.R1.fastq.gz'
           R2_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L005_001.R2.fastq.gz'
           optional:
             library_id: 'libA'
             sequencing_platform: 'ILLUMINA'
         RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L006_001:
           R1_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L006_001.R1.fastq.gz'
           R2_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L006_001.R2.fastq.gz'
           optional:
             library_id: 'libA'
             sequencing_platform: 'ILLUMINA'
         RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L007_001:
           R1_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L007_001.R1.fastq.gz'
           R2_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L007_001.R2.fastq.gz'
           optional:
             library_id: 'libA'
             sequencing_platform: 'ILLUMINA'
         RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L008_001:
           R1_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L008_001.R1.fastq.gz'
           R2_fq.gz: '/data/kentsis/WGS/RPE-GFP-PGBD5/RPE-GFP-PGBD5-xeno_ATTCAGAA_HGFVTCCXX_L008_001.R2.fastq.gz'
           optional:
             library_id: 'libA'
             sequencing_platform: 'ILLUMINA'

   bam_inputs:
     RPE-GFP:
       bam_file: '/data/kentsis/WGS/Sample_RPE-GFP/xeno_RPE_GFP.mem.psorted.bam'
       bai_file: '/data/kentsis/WGS/Sample-RPE-GFP/xeno_RPE_GFP.mem.psorted.bai'
       read_groups:
         - '@RG     ID:RPE-GFP_1        PL:illumina     PU:Unknown      LB:RPE-GFP      DS:RefVersion:GRCh37.63-Sequencer:Unk
nown        SM:RPE-GFP      CN:NYGenome'
       pre-processing_already_complete: false
       matched_sample_params:
         is_matched_sample: true
         tumor_or_normal: 'normal'
```

An illustration of how the parameters defined in user_defined_workflow affects pipeline execution:



```
user_defined_workflow:  # Please specify which PG2 modules you wish to execute

  # The purpose of the genome personalization module is to enable PG2 to be genomic variant-awa
re. To achieve this, a diploid, organism-specific genome (sequence FASTA + annotation GTF) is (
enerated and subsequently utilized in various steps of proteome generation.
  # Required inputs are either DNA sequencing (WGS/WES) .fastq or .bam files, or a pre-made var
iant call file (.vcf).
  # Depending on input type, data pre-processing and/or variant calling should be enabled.
  genome_personalization_module:
    enable_genome_personalization_module: true

    # * If enable_genome_personalization_module == false, all of its submodules are automatica
ly disabled and parameters automatically ignored. *

    variant_calling_submodule:
      run_pre-processing_steps: true # true unless input is an analysis-ready bam (as defined b
y GATK Best Practices).
      call_germline_variants_with_GATK4_HaplotypeCaller: true # genotyping + VQSR will also be
performed

      if_inputs_are_matched_tumor-normal_samples:
        call_somatic_variants_with_GATK4_Mutect2: true # if true, somatic variants will be cal
ed on the paired sample

      #somatic variant calling on unpaired samples is currently unsupported

  RNA-seq_module:

    enable_RNA-seq_module: true # this module should be enabled whenever inputs include RNA-seq
 data, and disabled otherwise

    # when the transcriptome assembly submodule is invoked, genes will be represented in the ou
tput protein database if and only if they are expressed, UNLESS merged with a gene set from the
 (stock or organism-specific) genome annotation
    transcriptome_track:
      assemble_transcriptome_with_StringTie: true
      GTF-guided-mapping_or_denovo-assembly: 'denovo' # {'guided', 'denovo'}

    gene_fusion_track:
      assemble_fusion_genes_with_STAR-Fusion: false

  # Create a genome track that propagates gene subset(s) from the genome annotation GTF direct
y into protein prediction. Importantly, this is distinct from, and does not impact rules that (
```

*

```
(snakemake) [kwokn@lilac 1-24_pgm2]$ snakemake --snakefile 1-28_main.py --
100 -k --ri --latency-wait 30 --configfile 1-30_testMain.yaml out/combined
Job counts:
        count   jobs
        1       pre_00a_CreateIntervalsWithoutNs
        1       pre_00b_ScatterBaseRecalibratorIntervals
        5       pre_01_GenerateRGWiseUbamsFromFq
        5       pre_02_AlignAndSortReadsByRG_FQ
        5       pre_03_MergeMappedAndUnmappedReadsByRG_FQ
        2       pre_04_SambambaMergeAllRGs
        2       pre_05_SambambaMarkdups
        2       pre_06_SortAndFixRealignedBamTags
        40      pre_07_BaseRecalibrator
        2       pre_08_GatherBqsrReports
        40      pre_09a_ApplyBQSRToMappedReads
        2       pre_09b_ApplyBQSRToUnmappedReads
        2       pre_10_GatherRecalibratedBAMs
        109
Job counts:
        count   jobs
        50      var_00_ScatterVariantCallingIntervals
        2       var_00_SymlinkToPreProcessingOutputBam
        100     var_germ_01_CallGermlineVariantsPerInterval
        50      var_germ_02_CombineSampleGVCFsPerInterval
        50      var_germ_03_GenotypeJointCohortPerInterval
        1       var_germ_04_MergeIntervalWiseJointCohortGenotypedVCFs
        1       var_germ_05_CalculateVQSR_INDEL
        1       var_germ_05_CalculateVQSR_SNP
        2       var_germ_06_ApplyVQSR
        1       var_germ_07_MergeRecalibratedVCFs
        1       var_som_00a_GetTumorPileupSummaries
        1       var_som_00b_CalculateContamination
        50      var_som_01_Mutect2_matched_tumor_normal
        1       var_som_02_MergeScatteredMutect2VCFs
        1       var_som_02a_MergeScatteredMutect2Stats
        1       var_som_03_FilterMutectCalls
        1       var_z_MergeFinishedVCFs
        314
Job counts:
        count   jobs
        910     genome_01_CreateChrWiseCustomRef
        2       genome_02a_MergeChrWiseCustomRef
        2       genome_02b_LiftoverAnnotationGTF
        914
Job counts:
        count   jobs
        2       RNA_00_STAR_CreateGenomeIndex
        6       RNA_01_STAR_AlignRNAReadsByRG
        6       RNA_02_FilterLowQualityReads
```

Usage

Screen

Since most runs of PG2 will not be completed within 1 sitting, it is highly recommended that PG2 be run within *screen*, which ensures safety and continuation of a run even if the connection to the head node is lost:

```
screen -S pg2
```

In the event of a lost connection, the screen can be retrieved with `screen -r pg2`.

Activating Singularity

```
module load singularity/3.6.0
```

Basic Command

The snakemake command to run PG2 takes the following general form:

```
snakemake --snakefile ProteomeGenerator2.py --cluster "bsub -J {params.J} -n
{params.n} -R 'span[hosts=1] rusage[mem={params.mem_per_cpu}]' -W 144:00 -o {params.o}
-eo {params.eo}" -j 100 -k --ri --latency-wait 30 \
--configfile configfiles/PG2_template.yaml --use-conda --use-singularity --
singularity-args "--bind /data:/data,/lila:/lila,/scratch:/scratch" -n --quiet
```

The command admittedly looks really complicated at first, but keep in mind that most of it doesn't change from run to run, so you can pretty much copy & paste it for each run, with minor modifications. A brief rundown of the parameters:

<u>Quality checks recommended before every pipeline calculation:</u>

-n = test run. Lists out every pipeline rule that will be executed with the given command. Without --quiet, includes specification of inputs, outputs, and "wildcard" parameters. Extremely useful sanity check, highly recommend running this before every run.

--quiet = reduces the verbosity of the run/dry-run terminal output to print only the names of the rules being run, without input/output/wildcards. Suggested to include in dry-run command before kicking off a full pipeline run (as command list can be quite unwieldy with subworkflows etc.) but would remove prior to the actual run.

<u>Parameters that user will change regularly</u>:

--configfile <configfile.yaml> = where the workflow is defined, and all directories/files/parameters are set

<output_file> = the 'target', or final output of the pipeline desired by the user

--snakefile <snakefile.py> = the main snakemake (Python) program to execute (for PG2, this is ProteomeGenerator2.py) → this will vary only if the user wants to run the various submodules (pre-processing, variant calling, etc) independently of the main program

Parameters that the user will likely have to change once before first execution

--cluster "<cluster_specific_string>" = parameters for compute cluster execution (the Lilac cluster uses the LSF scheduler, which deploys jobs via the command *bsub*). {params.<parameter>} binds the corresponding parameters of the *params section* of each Snakemake rule, to the cluster command. *This will need to be edited for use with SLURM, Oracle Grid Engine, etc.* (here, J=jobname, n=cores, R=host/memory spec, W=wall time, o=output log, eo=error log)

--singularity-args "--bind <filesystem_bindings>" = grants Singularity containers permission to access other directories within the local filesystem

Parameters the user will likely (and probably should) never change:

--use-conda = allows use of packages and environments from *Anaconda*

--use-singularity = allows use of plug-n-play containers (PG2 uses Singularity specifically for MaxQuant, which requires particular versions of *mono* to run on Linux)

-k = when a job fails, continue executing independent jobs

--ri = rerun incomplete jobs automatically

For the full parameter specifications, please see the *Snakemake* official documentation at https://snakemake.readthedocs.io/en/stable/.

Output files

The usage of PG2 follows the general paradigm of *Snakemake,* in which the user specifies, in the snakemake command, the **final output file(s)** they wish to produce. Snakemake then identifies the rule (every pipeline step is defined by a *rule*) that produces the target output file, and then *works backwards* to string together the set of rules (represented by a DAG) needed to generate the output files, until reaching the input files.

Default case: If no particular output file(s) are specified, then *Snakemake* will default to the files specified in the "all" rule of the Snakefile in question (for PG2, the Snakefile will typically be ProteomeGenerator2.py). Here are the default output files specified in rule *all*:

- Out/{study_group}/**combined.proteome.unique.fasta** : the primary output proteome database. This is a non-redundant (unique) set of customized protein sequences, which is created by combining and de-duplicating the databases from the parallel haplotype runs (which themselves are located at out/{study_group}/haplotype-{1,2}/transcriptome/proteome.fasta). This combined database is what is used as the search database for the proteomics module.

- Out/{study_group}/**combined.proteome.bed**: the BED file corresponding to the aforementioned proteome database, with coordinates lifted *back* to the original stock genome coordinate space (for easy visualization in IGV). The bedfiles in custom genome coordinate space for each haplotype run are located at /out/{study_group}/haplotype-{1,2}/transcriptome/proteome_preLiftBack.bed.

- Out/{study_group}/MaxQuant/combined/txt/**summary.txt** : the significance of this file is that its presence signifies the completion of a MaxQuant run. The main output tables such as *peptides.txt*, *proteinGroups.txt*, *msms.txt*, etc. are also contained within the same directory. *Summary.txt* itself contains various metrics about the run, both separated by raw file and in aggregate.

- Out/{study_group}/novel_analysis/{mutation_type}/**combined.{mutation_type}.map** : Map of all variants of a given mutation type that appear in the proteome database. Thus far this has been implemented for {missense, insertions, deletions, frameshifts}. *Additionally*, combined.{mutation_type}_MQevidence.map maps mutations to MaxQuant (MQ) evidence, and combined.novelPep_{mutation_type}.map maps MQ-identified non-canonical peptides to their originating mutations.

Authors

Nathaniel Kwok, Zita Aretz and Alex Kentsis