# Varathon

# User Manual

## Varathon: a scalable variant calling and benchmarking framework for both short and long reads

Release v1.0.0beta
(2019-05-10)

Author Contact:

Jia-Xing Yue (岳家兴)

Email: yuejiaxing@gmail.com
GitHub: yjx1217
Twitter: iAmphioxus
Website: http://www.iamphioxus.org

# Table of Contents

## Introduction

Varathon is a scalable variant calling and benchmarking framework that supports both short and long reads. In addition to small variants such as SNPs and INDELs, Varathon can also identify large variants such as structural variants (SVs) (e.g. inversions, translocations, segmental deletions and duplications). Copy-number variants (CNVs), the results of segmental deletions and duplications, can also be nicely profiled. Under the hood, a series of task-specific modules are provided to carry out the full workflow of read-mapping-based variant calling:

- **00. Reference_Genomes**
  preparing reference genome(s)
- **00.Simulated_Genomes**
  generating simulated genome(s) (by simuG)
- **00.Short_Reads**
  downloading (by SRA tools) or simulating (by ART) short reads
- **00.Long_Reads**
  downloading (by SRA tools) or simulating (by SimLoRD or DeepSimulator) long reads
- **01.Short_Read_Mapping**
  mapping short-read mapping to the reference genome (by bwa)
- **02.Short_Read_SNP_INDEL_Calling**
  calling SNPs and INDELs based on the short-read mapping alignment (by GATK4 or freebayes)
- **03.Short_Read_SV_Calling**
  calling SVs based on the short-read mapping alignment (by Manta or Delly)
- **04.Short_Read_CNV_Calling**
  calling CNVs based on the short-read mapping alignment (by Freec+DNAcopy)
- **11.Long_Read_Mapping**
  mapping long reads to the reference genome (by minimap2, pbmm2, ngmlr, or last)
- **12.Long_Read_SNP_INDEL_Calling**
  calling SNPs and INDELs based on the long-read mapping alignment (by longshot)
- **13.Long_Read_SV_Calling**
  calling SVs based on the long-read mapping alignment (by Sniffles, svim, Picky, nanosv, or pbsv)
- **20.Variant_Calling_Benchmarking**
  comparing different variant calling VCF files to calculate benchmarking statistics such as precision, recall, and F1 score.

## Citation

Manuscript in preparation.

## License

## Installation

Varathon is implemented in Bash, Perl5 and R. It is designed for a desktop or computing server running an x86-64-bit Linux operating system. Multithreaded processors are preferred to speed up the process since many steps can be configured to use multiple threads in parallel.

```
git clone https://github.com/yjx1217/Varathon.git
cd Varathon
bash ./install_dependencies.sh
```

## What's Inside

Inside the downloaded Varathon directory, you should see the following file structure:

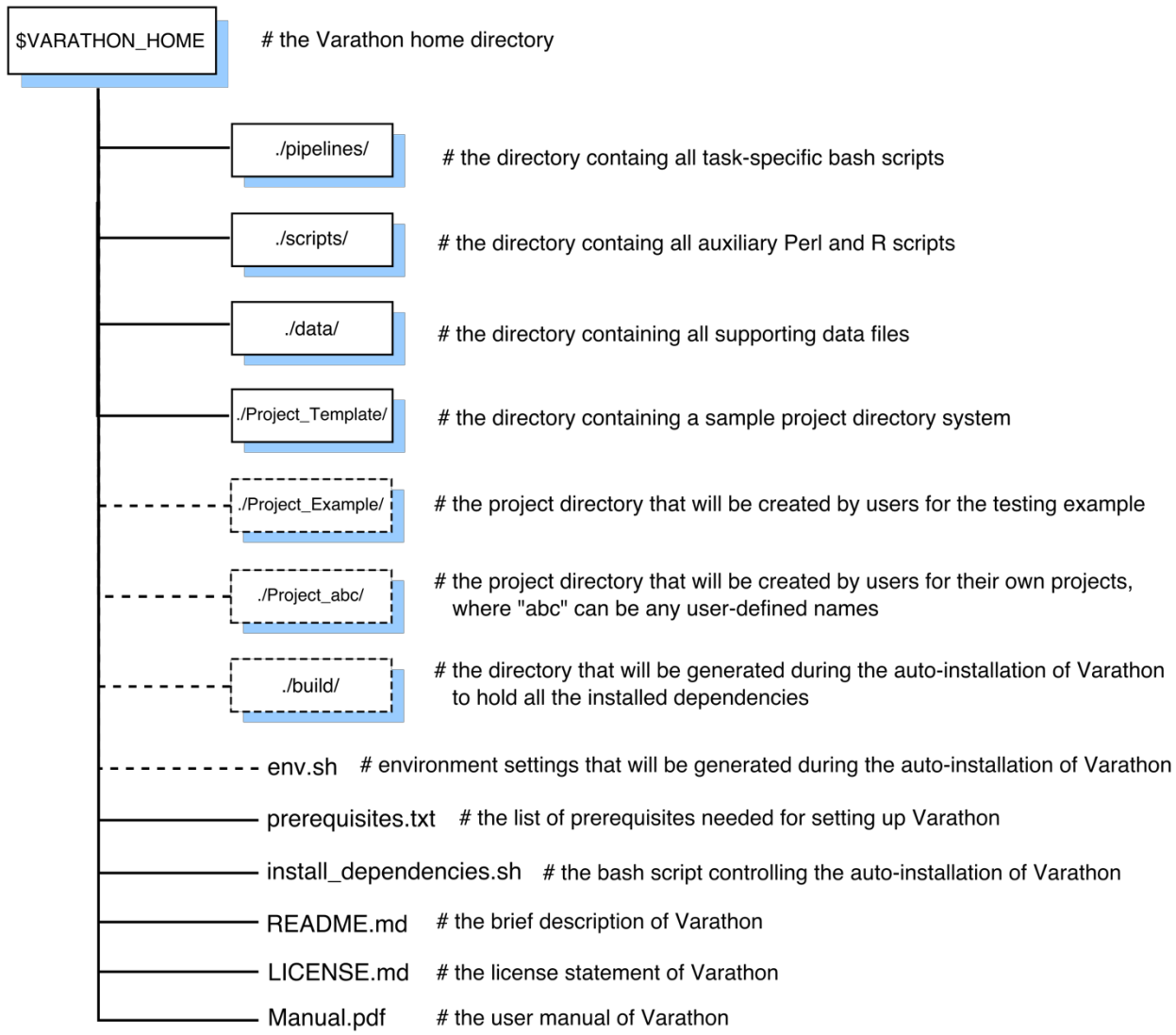


***Figure 1. Overview of the Varathon directory system.*** *All top-level directories (boxes, solid lines) and individual files of Varathon are listed and briefly described. Additional directories and files will be generated during the installation and execution of Varathon (boxes, dashed lines).*

## Pipeline Designs

With Varathon, we designed a highly structured project directory system to help users to perform their variant calling analysis in an organized and modular fashion (Figure 2). Within such project directory system, four subdirectories numbered as “00” are used for holding the reference and simulated genome(s) as well as the real and simulated short- (Illumina) and long- (PacBio and Nanopore) reads. The task-specific subdirectories for short-read mapping and different types (SNP/INDEL, SV, and CNV) of variant calling are numbered sequentially from “01” to “04”. Their counterparts relied on long-reads are numbered from “11” to “13”. Finally, a dedicated subdirectory numbered as “20” is used for comparing the ground truth with the calling results (in VCF format) from different callers. For read mapping and variant calling (module 01-04 and 11-13), a simple tab-delimited master sample table is used for specifying all the samples in the same processing batch and all those modules will ran through all the samples accordingly, making the whole process highly scalable.

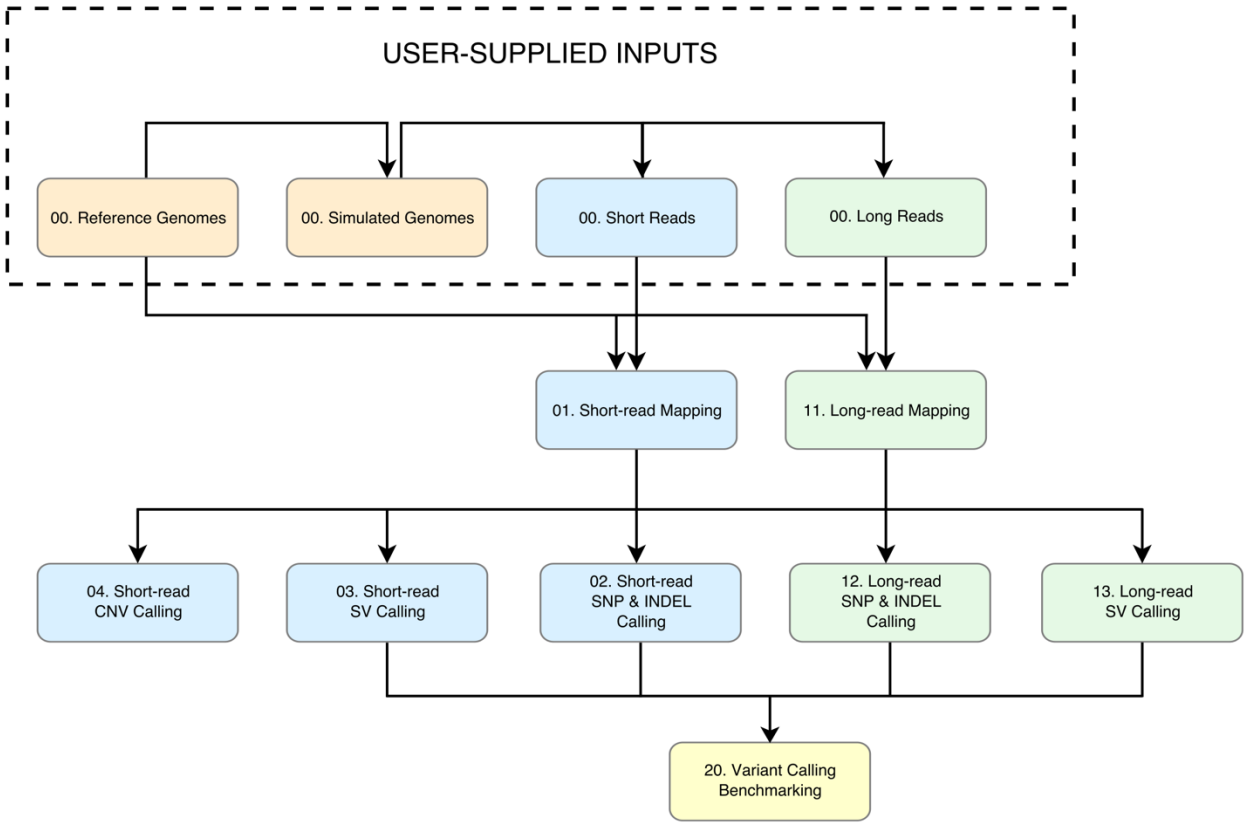


***Figure 2. The workflow of Varathon.*** *Each box represents an individual module. These modules are numbered according to the tasks described in Introduction.*

## Testing Example Working Through

1. Downloading and installing Varathon:

```
git clone https://github.com/yjx1217/Varathon.git
cd Varathon
bash ./install_dependencies.sh
```

Please note that it will take ~45 min for the installation to finish. Therefore, it is recommended to run the bash script above with nohup, which prevents the unintended interruption of the running script:

```
nohup bash ./install_dependencies.sh > run.log.txt 2>&1 &
```

If the installation succeeds, you are expected to see the following massage that signify the successful of executing this bash script. The same is true for all the bash scripts shipped with Varathon:

Varathon message: This bash script has been successfully processed! :)

2. Copying the Project_Template directory to create your own Varathon project directory. Let's name it as Project_Example for this testing example. Once created, enter into this directory.

```
cp -r Project_Template Project_Example
cd Project_Example
```

3. Setting up the reference genome for our testing example. For this example, we are going to use the budding yeast *Saccharomyces cerevisiae* reference genome R64-1-1.

```
cd 00.Reference_Genomes
bash Varathon.00.Prepare_Reference_Genome.sh
```

By running this bash script, the yeast reference genome will be automatically downloaded from the Ensembl database and be properly set up for our downstream analysis. During this process, chromosome/scaffold/contig names will be simplified by removing anything after the first space (including the space). Also, users can easily exclude any chromosome(s)/scaffold(s)/contig(s) by defining a single-column list file (specified by the option excluded_chr_list in the bash script). For our testing example, we excluded the mitochondrial genome "chrMT" from the yeast reference genome. For your own project, you can use this script to prepare your own reference genome by customizing the options for genome prefix, downloading URL, and exclusion list.

4. Setting up the simulated genome for our testing example. For our testing example, we will take the yeast reference genome as the template to simulate one yeast genome with 10000 random SNPs (denoted as "yeast.SNP" hereafter) and another with 10 segmental deletions (denoted as "yeast.CNV_DEL" hereafter).

```
cd ./../00.Simulated_Genomes
bash Varathon.00.Simulated_Genome.sh
```

The pre-set options in this bash script will simulate the yeast.CNV_DEL genome for you. To simulate the yeast.SNP genome, set the option output_prefix to "yeast.SNP" and set the option additional_simuG_options to "snp_count=10000;titv_ratio = 0.5" and then run this bash script. For generating more simulated genomes, please check simuG (https://github.com/yjx1217/simuG) for all supported options.

5. Setting up short reads. For this testing example, we will simulate Illumina reads based on the two simulated genomes that we generated above.

```
cd ./../00.Short_Reads
bash Varathon.00.Simulated_Short_Reads.sh
```

The pre-set options in this bash script will simulate 30x 150-bp Illumina paired-end reads for the yeast.CNV_DEL simulated genome. Modify the option ref_genome and output_prefix and then ran this script again to simulate the reads for the yeast.SNP simulated genome. For your own project, you can simply put your own Illumina paired-end reads (in *.fastq.gz format) in this directory. We also provide a bash script Varathon.00.Retrieve_Short_Reads_from_SRA.sh in case you want to download Illumina reads from NCBI SRA database. A simple SRA run ID list file needs to be provided to use this script. We provide such an example: ./../../data/yeast.short_reads.sra_run_list.txt. In this file, the first column is mandatory, which specifies the SRA run IDs for all the reads that you want to download. All lines started with "#" will be ignored in this file.

6. Setting up long reads. For this testing example, we will simulate long reads based on the two simulated genomes as well.

```
cd ./../00.Long_Reads
bash Varathon.00.Simulated_Long_Reads.sh
```

The pre-set options in this bash script will simulate the 30x PacBio reads for the yeast.CNV_DEL simulated genome. Like what we did for simulating short reads, modify the option ref_genome and output_prefix and then ran this script again to simulate the reads for the yeast.SNP simulated genome. The same script can also be used for simulating Nanopore reads. For your own project, you can put your own PacBio or Nanopore reads (in *.fastq.gz format) in this directory. In addition, two other bash scripts are further provided for downloading long reads from SRA database and for filtering/downsampling/cleaning long reads. The options enclosed in these scripts should be self-explanatory.

7. Mapping short-reads to the reference genome. As stated in the pipeline design section, Varathon use a tab-delimited master sample table to control read mapping and variant calling in a batch by batch fashion. Such master sample table should contain three columns: sample_id, read_file_name, and user_notes. The first two columns are mandatory and the third column is only for user's self-recording. All lines started with "#" will be automatically ignored. For short-read mapping and variant calling modules, Varathon will assume the paired-end Illumina reads are in the 00.Short_Reads directory. So only the file name is needed. A comma (",") is used to separate the forward (*.R1.fastq.gz) and reverse (*.R2.fstq.gz) paired-end reads. For the testing example, in the 00.Short_Read_Mapping subdirectory, we have prepared such a master sample table (Master_Sample_Table.Batch_yeast_sim_illumina.txt), in which our simulated short reads

have been specified. Run the corresponding task-specific bash script to start short read mapping. Upon the completion of short-read-mapping, a summary file (all_samples.coverage_summary.txt) will be generated in the batch-specific output directory (Batch_yeast_sim_illumina for the testing example) recording the mean and median mapping coverages for both the whole genome and each individual chromosome.

```
cd ./../01.Short_Read_Mapping
bash Varathon.01.Short_Read_Mapping.sh
```

8. Short-read-mapping-based SNP and INDEL calling. Same as above, we use the pre-defined master sample table to specify a batch of samples need to be processed. For short-read-mapping-based SNP and INDEL calling, Varathon supports two popular callers: GATK4 and freebayes. For the testing example, type:

```
cd ./../02.Short_Read_SNP_INDEL_Calling
bash Varathon.02.Short_Read_SNP_INDEL_Calling.sh
```

9. Short-read-mapping-based SV calling. Same as above, we use the pre-defined master sample table to specify a batch of samples need to be processed. For short-read-mapping-based SV calling, Varathon supports two popular callers: Manta and Delly. For the testing example, type:

```
cd ./../03.Short_Read_SV_Calling
bash Varathon.03.Short_Read_SV_Calling.sh
```

10. Short-read-mapping-based CNV calling. As a special case of SVs, CNVs can also be efficiently profiled using a sliding-window-based genomic scan based on mapping coverage, so Varathon also provides such a module as a bonus. For short-read-mapping-based CNV calling, Varathon uses a combination of FREEC and DNAcopy to scan across the genome. Same as above, we use the pre-defined master sample table to specify a batch of samples need to be processed. For the testing example, type:

```
cd ./../04.Short_Read_CNV_Calling
bash Varathon.04.Short_Read_CNV_Calling.sh
```

11. Mapping long-reads to the reference genome. Varathon supports four long-read mappers: minimap2, ngmlr, last, and pbmm2. Based on our experiences, their processing speed goes as follows: pbmm2 > minimap2 > ngmlr > last. For the testing example, we are going to use the default long-read mapper: minimap2. Like before, we use a master sample table to specify all the samples in the same processing batch. For our testing example, a pre-defined master sample table (Master_Sample_Table.Batch_yeast_sim_pacbio.txt) is provided. Like short-read-mapping, a summary file will also be generated in the batch-specific output directory to summarize the mapping coverage information for each processed sample.

```
cd ./../11.Long_Read_Mapping
bash Varathon.11.Long_Read_Mapping.sh
```

12. Long-read-mapping-based SNP and INDEL calling. Same as above, we use a master sample table to specify a batch of samples need to be processed. For our testing example, a pre-defined master sample table (Master_Sample_Table.Batch_yeast_sim_pacbio.txt) is provided.

For long-read-mapping-based SNP and INDEL calling, Varathon supports the only such caller in the field: longshot. For the testing example, type:

```
cd ./../12.Long_Read_SNP_INDEL_Calling
bash Varathon.02.Long_Read_SNP_INDEL_Calling.sh
```

13. Short-read-mapping-based SV calling. Same as above, we use a master sample table to specify a batch of samples need to be processed. For our testing example, a pre-defined master sample table (Master_Sample_Table.Batch_yeast_sim_pacbio.txt) is provided. For long-read-mapping-based SV calling, Varathon supports five popular callers: Sniffles, svim, Picky, nanosv, and pbsv, in which pbsv only supports long-read mapping alignments generated by pbmm2. For the testing example, we will use Sniffles. Running this module by typing:

```
cd ./../13.Long_Read_SV_Calling
bash Varathon.13.Long_Read_SV_Calling.sh
```

14. Comparing ground truth with called variants for benchmarking. Specify the reference and query VCF files (in *.vcf or *.vcf.gz format) in the task-specific bash script and run the script by:

```
cd ./../20.Variant_Calling_Benchmarking
bash Varathon.20.Variant_Calling_Benchmarking.sh
```