VAR-Seq Workflow Template

***Author:* Daniela Cassol (danielac@ucr.edu) and Thomas Girke (thomas.girke@ucr.edu)**

Last update: 21 November, 2019

Package

systemPipeR 1.21.0

Contents

1 Introduction

Users want to provide here background information about the design of their VAR-Seq project.

1.1 Background and objectives

This report describes the analysis of a VAR-Seq project studying the genetic differences among several strains . . . from *organism* . . . .

1.2 Experimental design

Typically, users want to specify here all information relevant for the analysis of their NGS study. This includes detailed descriptions of FASTQ files, experimental design, reference genome, gene annotations, etc.

2 Workflow environment

2.1 Generate workflow environment

Load workflow environment with sample data into your current working directory. The sample data are described here.

```
library(systemPipeRdata)
genWorkenvir(workflow = "varseq")
setwd("varseq")
```

Alternatively, this can be done from the command-line as follows:

```
Rscript -e "systemPipeRdata::genWorkenvir(workflow='varseq')"
```

In the workflow environments generated by `genWorkenvir` all data inputs are stored in a `data/` directory and all analysis results will be written to a separate `results/` directory, while the `systemPipeVARseq.Rmd` script and the `targets` file are expected to be located in the parent directory. The R session is expected to run from this parent directory. Additional parameter files are stored under `param/`.

To work with real data, users want to organize their own data similarly and substitute all test data for their own data. To rerun an established workflow on new data, the initial `targets` file along with the corresponding FASTQ files are usually the only inputs the user needs to provide.

2.2 Run workflow

Now open the R markdown script `systemPipeVARseq.Rmd`in your R IDE (*e.g.* vim-r or RStudio) and run the workflow as outlined below.

Here pair-end workflow example is provided. Please refer to the main vignette `systemPipeR.Rmd` for running the workflow with single-end data.

2.2.1 Run R session on computer node

In a computer cluster enviornment. Typically, after opening the `Rmd` file of this workflow in Vim and attaching a connected R session via the `F2` (vim-r plugin installed) key, following command sequence can be used to run your R session on a computer node.

```
q("no")  # closes R session on head node
```

```
srun --x11 --partition=short --mem=2gb --cpus-per-task 4 --ntasks 1 --time 2:00:00 --pty bash -l
module load R/3.6.0
R
```

Now check your R session running environment.

```
system("hostname")  # should return the computer name or cluster name
getwd()  # checks current working directory of R session
dir()  # returns content of current working directory
```

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (H Backman and Girke 2016).

```
library(systemPipeR)
## Loading required package: Rsamtools
## Loading required package: GenomeInfoDb
## Loading required package: BiocGenerics
## Loading required package: parallel
##
## Attaching package: 'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall,
##     clusterEvalQ, clusterExport, clusterMap,
##     parApply, parCapply, parLapply, parLapplyLB,
##     parRapply, parSapply, parSapplyLB
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename,
##     cbind, colnames, dirname, do.call, duplicated,
##     eval, evalq, Filter, Find, get, grep, grepl,
##     intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, Position, rank, rbind, Reduce,
##     rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max,
##     which.min
## Loading required package: S4Vectors
## Loading required package: stats4
##
## Attaching package: 'S4Vectors'
## The following object is masked from 'package:base':
```

```
##
##     expand.grid
## Loading required package: IRanges
## Loading required package: GenomicRanges
## Loading required package: Biostrings
## Loading required package: XVector
##
## Attaching package: 'Biostrings'
## The following object is masked from 'package:base':
##
##     strsplit
## Loading required package: ShortRead
## Loading required package: BiocParallel
## Loading required package: GenomicAlignments
## Loading required package: SummarizedExperiment
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view
##     with 'browseVignettes()'. To cite Bioconductor,
##     see 'citation("Biobase")', and for packages
##     'citation("pkgname")'.
## Loading required package: DelayedArray
## Loading required package: matrixStats
##
## Attaching package: 'matrixStats'
## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
##
## Attaching package: 'DelayedArray'
## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins,
##     rowRanges
## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum
##
##
## Attaching package: 'systemPipeR'
## The following object is masked from 'package:BiocStyle':
##
##     output
```

If applicable users can load custom functions not provided by `systemPipeR`. Skip this step if this is not the case.

```
source("systemPipeVARseq_Fct.R")
```

If you are running on a single machine, use following code as an exmaple to check if some tools used in this workflow are in your environment **PATH**. No warning message should be shown if all tools are installed.

3 Read preprocessing

3.1 Experiment definition provided by `targets` file

The `targets` file defines all FASTQ files and sample comparisons of the analysis workflow.

```
targetspath <- system.file("extdata", "targetsPE.txt", package = "systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")
targets[1:4, 1:4]
##                      FileName1                   FileName2
## 1 ./data/SRR446027_1.fastq.gz ./data/SRR446027_2.fastq.gz
## 2 ./data/SRR446028_1.fastq.gz ./data/SRR446028_2.fastq.gz
## 3 ./data/SRR446029_1.fastq.gz ./data/SRR446029_2.fastq.gz
## 4 ./data/SRR446030_1.fastq.gz ./data/SRR446030_2.fastq.gz
##   SampleName Factor
## 1        M1A     M1
## 2        M1B     M1
## 3        A1A     A1
## 4        A1B     A1
```

3.2 Read quality filtering and trimming

The following removes reads with low quality base calls (here a certain pattern) from all FASTQ files.

```
targetsPE <- system.file("extdata", "targetsPE.txt", package = "systemPipeR")
dir_path <- system.file("extdata/cwl/preprocessReads/trim-pe",
    package = "systemPipeR")
trim <- loadWorkflow(targets = targetsPE, wf_file = "trim-pe.cwl",
    input_file = "trim-pe.yml", dir_path = dir_path)
trim <- renderWF(trim, inputvars = c(FileName1 = "_FASTQ_PATH1_",
    FileName2 = "_FASTQ_PATH2_", SampleName = "_SampleName_"))
trim
output(trim)[1:2]

preprocessReads(args = trim, Fct = "trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)",
    batchsize = 1e+05, overwrite = TRUE, compress = TRUE)
writeTargetsout(x = trim, file = "targets_trimPE.txt", step = 1,
    new_col = c("FileName1", "FileName2"), new_col_output_index = c(1,
        2), overwrite = TRUE)
```

3.3 FASTQ quality report

The following `seeFastq` and `seeFastqPlot` functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The results are written to a PDF file named `fastqReport.pdf`. Use the output from previous step (fastq trimming) as the demonstration here to generate fastq report.

```
fqlist <- seeFastq(fastq = infile1(trim), batchsize = 1e+05,
    klength = 8)
pdf("./results/fastqReport.pdf", height = 18, width = 4 * length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

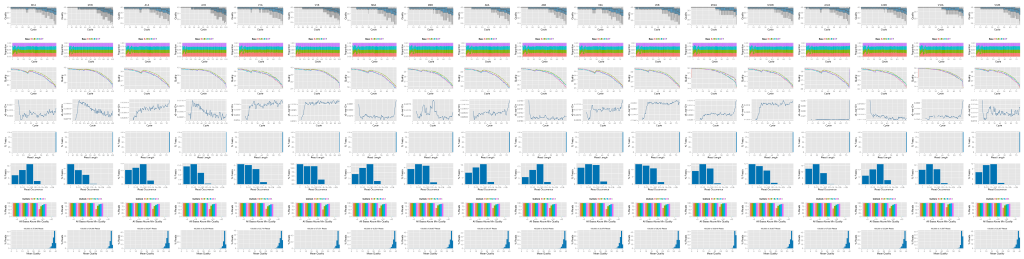

Figure 1: FASTQ quality report for 18 samples

4 Alignments

4.1 Read mapping with `BWA-MEM`

The NGS reads of this project are aligned against the reference genome sequence using the highly variant tolerant short read aligner `BWA-MEM` (Li 2013; Li and Durbin 2009). The parameter settings of the aligner are defined in the `gatk/bwa-pe.cwl`

DNA sequencing nowadays are usually solid for base quality and therefore, trimming is usually not needed in most cases. Also, variant calling tools like `GATK` will automatically not consider low quality bases. Therefore, this test code uses untrimmed fastqs. However, it is best to test with `FASTQ quality report` function provided above to verify on your real data first.

4.1.1 Build index and dictionary files for BWA and GATK

The following object `dir_path` is the folder where all `BWA` and `GATK` param files are located.

```
dir_path <- system.file("extdata/cwl/gatk", package = "systemPipeR")
```

Build the index and dictionary files for BWA and GATK to run.

```
## Index for BWA
args <- loadWorkflow(targets = NULL, wf_file = "bwa-index.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args)
cmdlist(args)  # shows the command
output(args)  # shows the expected output files
# Run single Machine
runCommandline(args, make_bam = FALSE)

## Index needed for gatk tools
```

```
args <- loadWorkflow(wf_file = "fasta_dict.cwl", input_file = "gatk.yaml",
    dir_path = dir_path)
args <- renderWF(args)
args <- runCommandline(args, make_bam = FALSE)

## Index
args <- loadWorkflow(wf_file = "fasta_faidx.cwl", input_file = "gatk.yaml",
    dir_path = dir_path)
args <- renderWF(args)
args <- runCommandline(args, make_bam = FALSE)
```

4.1.2 Run the read mapping

```
targetsPE <- system.file("extdata", "targetsPE.txt", package = "systemPipeR")
args <- loadWorkflow(targets = targetsPE, wf_file = "bwa-pe.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FASTQ_PATH1_",
    FileName2 = "_FASTQ_PATH2_", SampleName = "_SampleName_"))
cmdlist(args)[1:2]
output(args)[1:2]
```

Runs the alignments sequentially (*e.g.* on a single machine) by `runCommandline` function.

```
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args[1:2], file = "./results/targetsPE.txt",
    step = 1, new_col = "BWA_SAM", new_col_output_index = 1,
    overwrite = TRUE)
```

Alternatively, the alignment jobs can be submitted to a compute cluster. Here is the example to run cluster jobs by `clusterRun` on a `slurm` based system. 4 cpus for each task for 18 samples, totally 72 cpus.

```
library(batchtools)
resources <- list(walltime = 120, ntasks = 1, ncpus = 4, memory = 1024)
reg <- clusterRun(args, FUN = runCommandline, more.args = list(args = args,
    dir = FALSE, make_bam = FALSE), conffile = ".batchtools.conf.R",
    template = "batchtools.slurm.tmpl", Njobs = 18, runid = "01",
    resourceList = resources)
getStatus(reg = reg)
writeTargetsout(x = args, file = "./results/targetsPE.txt", step = 1,
    new_col = "BWA_SAM", new_col_output_index = 1, overwrite = TRUE)
```

Check whether all BAM files have been created.

```
outpaths <- subsetWF(args, slot = "output", subset = 1, index = 1)
file.exists(outpaths)
```

4.2 Read and alignment stats

The following generates a summary table of the number of reads in each sample and how many of them aligned to the reference.

```
read_statsDF <- alignStats(args = args)
write.table(read_statsDF, "results/alignStats.xls", row.names = FALSE,
    quote = FALSE, sep = "\t")
```

4.3 Create symbolic links for viewing BAM files in IGV

The symLink2bam function creates symbolic links to view the BAM alignment files in a genome browser such as IGV. The corresponding URLs are written to a file with a path specified under urlfile, here IGVurl.txt.

```
symLink2bam(sysargs = args, htmldir = c("~/.html/", "somedir/"),
    urlbase = "http://cluster.hpcc.ucr.edu/~tgirke/", urlfile = "IGVurl.txt")
```

5 Variant calling

The following performs variant calling with GATK and BCFtools on a single machine by run Commandline function can be used to run the variant calling with GATK and BCFtools for each sample sequentially. If a cluster is available, running in parallel mode on a compute cluster can be performed by clusterRun (McKenna et al. 2010; Li 2011). Typically, the user would choose here only one variant caller rather than running several ones.

Not all users have a cluster system, so here to demonstrate an example variant calling workflow, only single-machine commands are shown. For cluster jobs, please refer to previous steps like code for BWA as a template to run on the cluster.

5.1 Variant calling with GATK

The following steps are based on GATK 4.1.1.0 Best Practice. A targets file is needed to load samples to a SYSargs2 intance. There are 10 individual steps where the user can choose where to jump in and where to skip. All scripts are located at param/cwl/gatk. BQSR (Base Quality Score Recalibration) and VQSR (Variant Quality Score Recalibration) are very specific to a limited species like human, so this workflow does not support these steps.

5.1.1 Step1: fastq to ubam

Convert fastq files to bam files to prepare for the following step. It is very important to specific your sequencing platform, default is illumina. User need to change gatk_fastq2ubam.cwl if the platform is different. Platform information is needed for the variant caller in later steps to correct calling parameters.

```
dir_path <- system.file("extdata/cwl/gatk", package = "systemPipeR")
targets.gatk <- "./results/targetsPE.txt"  ## targets generated from BWA
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_fastq2ubam.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FASTQ_PATH1_",
    FileName2 = "_FASTQ_PATH2_", SampleName = "_SampleName_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args[1:2], make_bam = FALSE)
```

```
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "GATK_UBAM", new_col_output_index = 1,
    overwrite = TRUE)
```

5.1.2 Step2: Merge `bam` and `ubam`

This step merges a `bam` and `ubam` and creates a third `bam` file that contains alignment information and remaining information that was removed by the aligner like `BWA`. The removed information is essential for variant statistics calculation. Previous steps are recommended, but variant calling can still be performed without these steps.

```
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_mergebams.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(BWA_SAM = "_bwasam_", GATK_UBAM = "_ubam_",
    SampleName = "_SampleName_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "GATK_MERGE", new_col_output_index = 1,
    overwrite = TRUE)
```

5.1.3 Step3: Sort `bam` files by genomic coordinates

Sort `bam` files by genomic coordinates.

```
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_sort.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_",
    GATK_MERGE = "_mergebam_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "GATK_SORT", new_col_output_index = 1,
    overwrite = TRUE)
```

5.1.4 Step4: Mark duplicates

Mark PCR artifacts in sequencing. A `duplicate_metrics` file will also be produced by this step, but will not be used for the next step. This file is just for the user to check duplicates status summary.

```
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_markduplicates.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_",
    GATK_SORT = "_sort_"))
cmdlist(args)[1:2]
```

```r
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = c("GATK_MARK", "GATK_MARK_METRICS"),
    new_col_output_index = c(1, 2), overwrite = TRUE)
```

5.1.5 Step5: Fixing tags

Takes the `bam` from the last step and calculates the NM, MD, and UQ tags. These tags are important for variant calling and filtering. This step is recommended but can be skipped.

```r
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_fixtag.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_",
    GATK_MARK = "_mark_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "GATK_FIXED", new_col_output_index = 1,
    overwrite = TRUE)
```

Up till this step, sample preprocess is done. All analysis ready `BAM` files and their index `.bai` files are created. Individual and cohort calling by `HaplotypeCaller` is performed from the next step.

5.1.6 Step6: HaplotypeCaller `gvcf`

The `HaplotypeCaller` is running a **gvcf** mode in this step. G stands for ‘genomic’. The file not only contains variant sites information but also non-variant sites information; thus, at the following step, the cohort caller can use this information to validate the true variants.

```r
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_haplotypecaller.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_",
    GATK_FIXED = "_fixed_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "GVCF", new_col_output_index = 1, overwrite = TRUE)
```

5.1.7 Step7: Import all `gvcfs`

It is recommended to import all **gvcfs** to a TileDB database for fast cohort variant calling at the following step. Note: if you are working with non-diploid data, use `CombineGVCFs` function from `GATK` and change the `gvcf_db_folder` parameter in `param/cwl/gatk/gatk.yaml` to be your combined **gvcf** file.

```
# drop all *.g.vcf.gz files into results folder, make sure
# the tbi index is also there.
args <- loadWorkflow(targets = NULL, wf_file = "gatk_genomicsDBImport.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args)
cmdlist(args)
output(args)
args <- runCommandline(args = args, make_bam = FALSE)
```

5.1.8 Step8: Cohort calling of `gvcf`

Assess variants by information from all gvcfs. A collective vcf called `samples.vcf.gz` is created by default naming.

```
args <- loadWorkflow(targets = NULL, wf_file = "gatk_genotypeGVCFs.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args)
cmdlist(args)
output(args)
args <- runCommandline(args = args, make_bam = FALSE)
```

5.1.9 Step9: Cohort hard filter variants

VQSR is not included in this workflow. Variants are hard filtered together. See this Post for parameters for hard filtering. Change these settings in `param/cwl/gak/gatk_variantFiltration.sh` if needed.

```
args <- loadWorkflow(wf_file = "gatk_variantFiltration.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args)
cmdlist(args)
output(args)
args <- runCommandline(args = args, make_bam = FALSE)
```

5.1.10 Step10: Extract variant

After cohort calling, filtering, all variants for all samples are stored in one big file. Extract variants for each sample and save them separately (only variants that have passed the filters are stored).

```
targets.gatk <- "./results/targets_gatk.txt"
args <- loadWorkflow(targets = targets.gatk, wf_file = "gatk_select_variant.cwl",
    input_file = "gatk.yaml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_"))
cmdlist(args)[1:2]
output(args)[1:2]
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_gatk.txt",
    step = 1, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

5.2 Variant calling with `BCFtools`

The following runs the variant calling with `BCFtools`. This tool takes `BWA` aligned `BAM` files, sort, mark duplicates by `samtools` and finally call variants by `BCFtools`.

```
dir_path <- system.file("extdata/cwl/workflow-bcftools", package = "systemPipeR")
targetsPE <- "./results/targetsPE.txt"
args <- loadWorkflow(targets = targetsPE, wf_file = "workflow_bcftools.cwl",
    input_file = "bcftools.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(SampleName = "_SampleName_",
    BWA_SAM = "_SAM_"))
cmdlist(args[1])
output(args[1])
args <- runCommandline(args = args, make_bam = FALSE)
writeTargetsout(x = args, file = "./results/targets_bcf.txt",
    step = 5, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

Variant calling ends here. Downstream analysis starts from the next section.

5.3 Inspect VCF file

Scripts of downstream analysis are stored in `param/cwl/varseq_downstream`

```
dir_path <- system.file("extdata/cwl/varseq", package="systemPipeR")
```

VCF files can be imported into R with the `readVcf` function. Both `VCF` and `VRanges` objects provide convenient data structure for working with variant data (*e.g.* SNP quality filtering).

```
library(VariantAnnotation)
args <- loadWorkflow(targets = "./results/targets_gatk.txt",
    wf_file = "filter.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_"))
vcf <- readVcf(infile1(args)[1], "A. thaliana")
vcf
vr <- as(vcf, "VRanges")
vr
```

6 Filter variants

The function `filterVars` filters VCF files based on user definable quality parameters. It sequentially imports each VCF file into R, applies the filtering on an internally generated `VRanges` object and then writes the results to a new subsetted VCF file. The filter parameters are passed on to the corresponding argument as a character string. The function applies this filter to the internally generated `VRanges` object using the standard subsetting syntax for two dimensional objects such as: `vr[filter, ]`. The parameter files are stored under `param/cwl/varseq_downstream` as dummy cwl files. Please do not run them in `cwltool`. These files are used in the filtering steps, which helps to define the paths to the input and output VCF files that are stored in `SYSargs2` instances.

6.1 Filter variants called by `GATK`

The below example filters for variants that are supported by `>=x` reads and >=80% of them support the called variants. In addition, all variants need to pass `>=x` of the soft filters recorded in the VCF files generated by GATK. Since the toy data used for this workflow is very small, the chosen settings are unreasonabley relaxed. A more reasonable filter setting is given in the line below (here commented out).

There is already some cohort filtering in GATK step 10. Some additional hard filtering is provided here. Apply if you need or skip this step.

```r
dir_path <- system.file("extdata/cwl/varseq", package = "systemPipeR")
library(VariantAnnotation)
library(BBmisc)  # Defines suppressAll()
args <- loadWorkflow(targets = "./results/targets_gatk.txt",
    wf_file = "filter.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
filter <- "totalDepth(vr) >= 2 & (altDepth(vr) / totalDepth(vr) >= 0.8)"
# filter <- 'totalDepth(vr) >= 20 & (altDepth(vr) /
# totalDepth(vr) >= 0.8)'
suppressAll(filterVars(args, filter, varcaller = "gatk", organism = "A. thaliana"))
writeTargetsout(x = args, file = "./results/targets_filter_gatk.txt",
    step = 1, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

6.2 Filter variants called by `BCFtools`

The following shows how to filter the VCF files generated by `BCFtools` using similar parameter settings as in the previous filtering of the GATK results.

```r
args <- loadWorkflow(targets = "./results/targets_bcf.txt", wf_file = "filter.cwl",
    input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
filter <- "rowSums(vr) >= 2 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
# filter <- 'rowSums(vr) >= 20 &
# (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)'
suppressAll(filterVars(args, filter, varcaller = "bcftools",
    organism = "A. thaliana"))
writeTargetsout(x = args, file = "./results/targets_filter_bcf.txt",
    step = 1, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

Check filtering outcome for one sample

```r
length(as(readVcf(infile1(args)[1], genome = "Ath"), "VRanges")[,
    1])
length(as(readVcf(subsetWF(args, slot = "output", subset = 1,
    index = 1)[1], genome = "Ath"), "VRanges")[, 1])
```

7 Annotate filtered variants

The function `variantReport` generates a variant report using utilities provided by the `VariantAnnotation` package. The report for each sample is written to a tabular file containing genomic context annotations (*e.g.* coding or non-coding SNPs, amino acid changes, IDs of affected genes, etc.) along with confidence statistics for each variant. The CWL file `param/cwl/varseq_downstream/annotate.cwl` defines the paths to the input and output files which are stored in a `SYSargs2` instance.

7.1 Basics of annotating variants

Variants overlapping with common annotation features can be identified with `locateVariants`.

```
dir_path <- system.file("extdata/cwl/varseq", package = "systemPipeR")
library("GenomicFeatures")
args <- loadWorkflow(targets = "./results/targets_filter_gatk.txt",
    wf_file = "annotate.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
txdb <- loadDb("./data/tair10.sqlite")
vcf <- readVcf(infile1(args)[1], "A. thaliana")
locateVariants(vcf, txdb, CodingVariants())
```

Synonymous/non-synonymous variants of coding sequences are computed by the predictCoding function for variants overlapping with coding regions.

```
fa <- FaFile(normalizePath(file.path(args$yamlinput$data_path$path,
    args$yamlinput$ref_name)))
predictCoding(vcf, txdb, seqSource = fa)
```

7.2 Annotate filtered variants `GATK`

```
library("GenomicFeatures")
args <- loadWorkflow(targets = "./results/targets_filter_gatk.txt",
    wf_file = "annotate.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(normalizePath(file.path(args$yamlinput$data_path$path,
    args$yamlinput$ref_name)))
suppressAll(variantReport(args = args, txdb = txdb, fa = fa,
    organism = "A. thaliana"))
writeTargetsout(x = args, file = "./results/targets_report_gatk.txt",
    step = 1, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

7.3 Annotate filtered variants `bcftools`

```
library("GenomicFeatures")
args <- loadWorkflow(targets = "./results/targets_filter_bcf.txt",
    wf_file = "annotate.cwl", input_file = "varseq.yml", dir_path = dir_path)
```

```
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(normalizePath(file.path(args$yamlinput$data_path$path,
    args$yamlinput$ref_name)))
suppressAll(variantReport(args = args, txdb = txdb, fa = fa,
    organism = "A. thaliana"))
writeTargetsout(x = args, file = "./results/targets_report_bcf.txt",
    step = 1, new_col = "FileName1", new_col_output_index = 1,
    overwrite = TRUE)
```

View annotation result for single sample

```
read.delim(output(args)[[1]][[1]])[38:40, ]
```

8 Combine annotation results among samples

To simplify comparisons among samples, the `combineVarReports` function combines all variant annotation reports referenced in a `SYSargs2` instance (here `args`). At the same time the function allows to consider only certain feature types of interest. For instance, the below setting `filtercol=c(Consequence="nonsynonymous")` will include only nonsysynonymous variances listed in the `Consequence` column of the annotation reports. To omit filtering, one can use the setting `filtercol="All"`.

8.1 Combine results `GATK`

```
dir_path <- system.file("extdata/cwl/varseq", package = "systemPipeR")
args <- loadWorkflow(targets = "results/targets_report_gatk.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
combineDF <- combineVarReports(args, filtercol = c(Consequence = "nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_gatk.xls",
    quote = FALSE, row.names = FALSE, sep = "\t")
```

8.2 Combine results `bcftools`

```
args <- loadWorkflow(targets = "results/targets_report_bcf.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
combineDF <- combineVarReports(args, filtercol = c(Consequence = "nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_bcf.xls",
    quote = FALSE, row.names = FALSE, sep = "\t")
```

9 Summary statistics of variants

The `varSummary` function counts the number of variants for each feature type included in the anntation reports.

9.1 Summary of variants GATK

```
args <- loadWorkflow(targets = "./results/targets_report_gatk.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
varSummary(args)
write.table(varSummary(args), "./results/variantStats_gatk.xls",
    quote = FALSE, col.names = NA, sep = "\t")
```

9.2 Summary of variants bcf

```
args <- loadWorkflow(targets = "./results/targets_report_bcf.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
varSummary(args)
write.table(varSummary(args), "./results/variantStats_bcf.xls",
    quote = FALSE, col.names = NA, sep = "\t")
```

10 Venn diagram of variants

The venn diagram utilities defined by the systemPipeR package can be used to identify common and unique variants reported for different samples and/or variant callers. The below generates a 4-way venn diagram comparing four sampes for each of the two variant callers.

```
dir_path <- system.file("extdata/cwl/varseq", package = "systemPipeR")
## gatk
args <- loadWorkflow(targets = "results/targets_report_gatk.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
varlist <- sapply(names(subsetWF(args[1:2], slot = "output",
    subset = 1, index = 1)), function(x) as.character(read.delim(subsetWF(args[1:2],
    slot = "output", subset = 1, index = 1)[x])$VARID))
vennset_gatk <- overLapper(varlist, type = "vennsets")

## bcf
args <- loadWorkflow(targets = "./results/targets_report_bcf.txt",
    wf_file = "combine.cwl", input_file = "varseq.yml", dir_path = dir_path)
args <- renderWF(args, inputvars = c(FileName1 = "_FILE1_", SampleName = "_SampleName_"))
varlist <- sapply(names(subsetWF(args[1:2], slot = "output",
    subset = 1, index = 1)), function(x) as.character(read.delim(subsetWF(args[1:2],
    slot = "output", subset = 1, index = 1)[x])$VARID))
vennset_bcf <- overLapper(varlist, type = "vennsets")

pdf("./results/vennplot_var.pdf")
vennPlot(list(vennset_gatk, vennset_bcf), mymain = "", mysub = "GATK: red; BCFtools: blue",
    colmode = 2, ccol = c("red", "blue"))
dev.off()
```

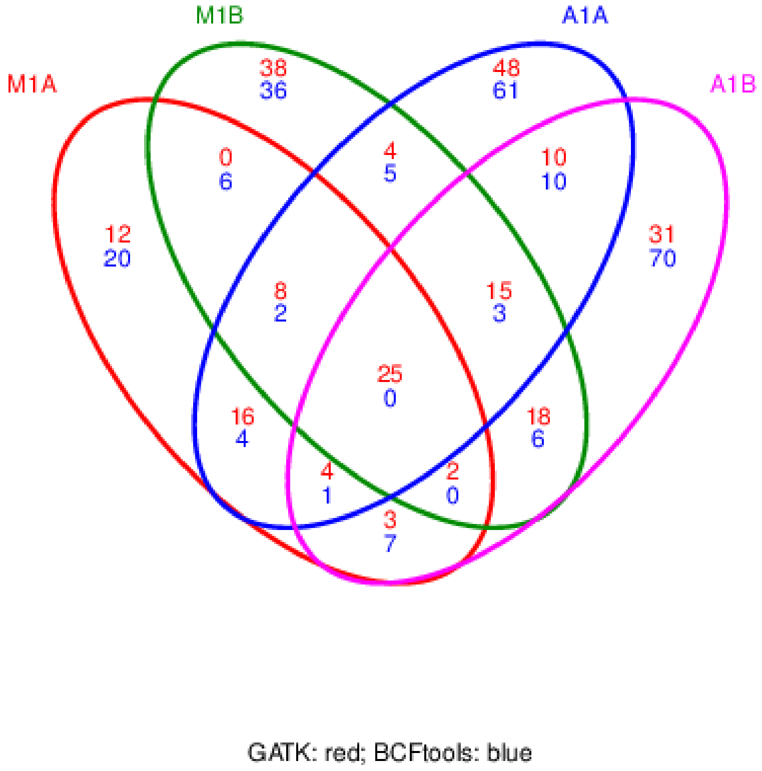

Figure 2: Venn Diagram for 4 samples from GATK and BCFtools

11 Plot variants programmatically

The following plots a selected variant with `ggbio`.

In this example, the input `BAM` file is from the `GATK` step 5, analysis ready bam. You can use other aligned `BAMs` as well, but make sure they are indexed. The `VCF` file is taken from `Inspect VCF file` section or you can load your own vcf.

```
library(ggbio)
mychr <- "ChrC"
mystart <- 11000
myend <- 13000
args <- loadWorkflow(targets = "results/targets_gatk.txt", wf_file = "combine.cwl",
    input_file = "varseq.yml", dir_path = "param/cwl/varseq_downstream/")
args <- renderWF(args, inputvars = c(GATK_FIXED = "_FILE1_",
    SampleName = "_SampleName_"))
ga <- readGAlignments(subsetWF(args, slot = "input", subset = 1)[1],
    use.names = TRUE, param = ScanBamParam(which = GRanges(mychr,
        IRanges(mystart, myend))))
p1 <- autoplot(ga, geom = "rect")
p2 <- autoplot(ga, geom = "line", stat = "coverage")
p3 <- autoplot(vcf[seqnames(vcf) == mychr], type = "fixed") +
    xlim(mystart, myend) + theme(legend.position = "none", axis.text.y = element_blank(),
    axis.ticks.y = element_blank())
p4 <- autoplot(loadDb("./data/tair10.sqlite"), which = GRanges(mychr,
    IRanges(mystart, myend)), names.expr = "gene_id")
png("./results/plot_variant.png")
tracks(Reads = p1, Coverage = p2, Variant = p3, Transcripts = p4,
    heights = c(0.3, 0.2, 0.1, 0.35)) + ylab("")
dev.off()
```

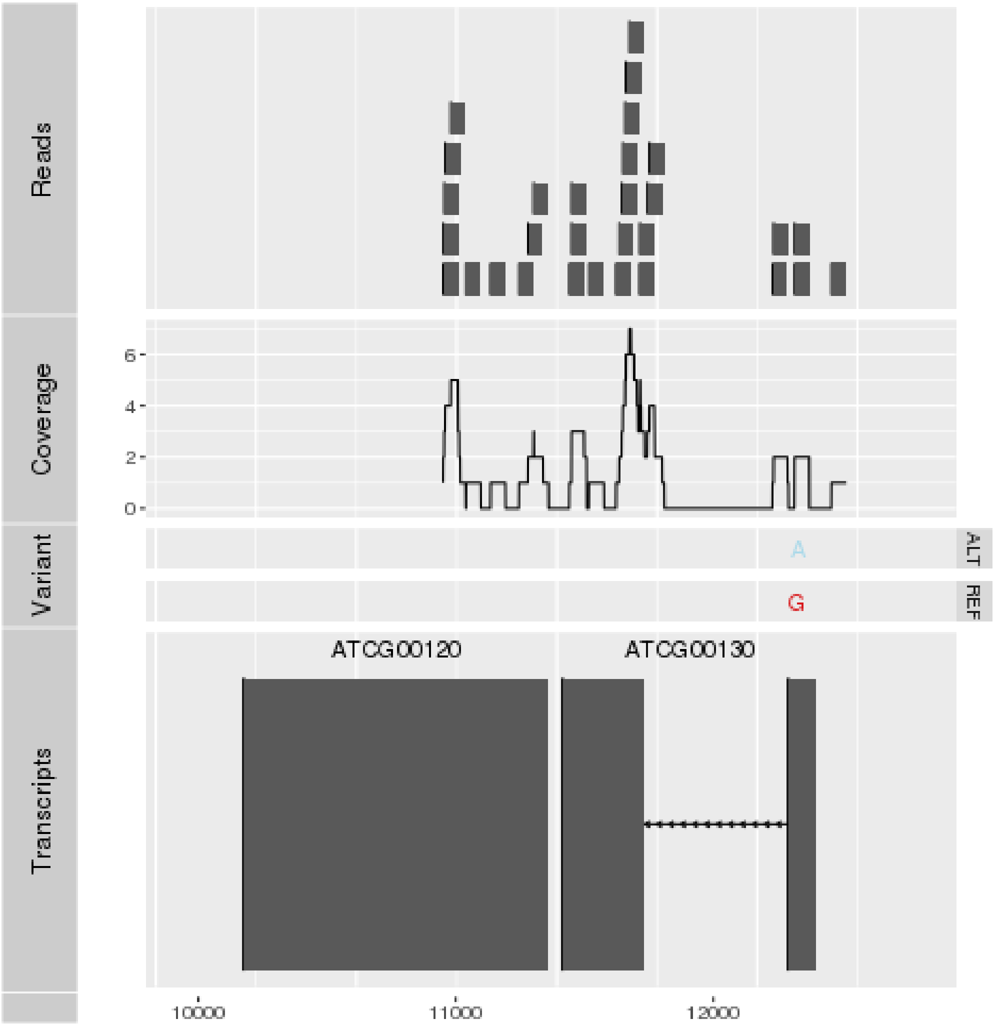

Figure 3: Plot variants with programmatically.

12 Version Information

```
sessionInfo()
## R version 3.6.1 (2019-07-05)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Pop!_OS 19.04
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices
## [6] utils     datasets  methods   base
##
## other attached packages:
##  [1] systemPipeR_1.21.0          ShortRead_1.44.0
##  [3] GenomicAlignments_1.22.1    SummarizedExperiment_1.16.0
##  [5] DelayedArray_0.12.0         matrixStats_0.55.0
##  [7] Biobase_2.46.0              BiocParallel_1.20.0
##  [9] Rsamtools_2.2.1             Biostrings_2.54.0
## [11] XVector_0.26.0              GenomicRanges_1.38.0
## [13] GenomeInfoDb_1.22.0         IRanges_2.20.1
## [15] S4Vectors_0.24.0            BiocGenerics_0.32.0
## [17] BiocStyle_2.14.0
##
## loaded via a namespace (and not attached):
##  [1] Category_2.52.1          bitops_1.0-6
##  [3] bit64_0.9-7              RColorBrewer_1.1-2
##  [5] progress_1.2.2           httr_1.4.1
##  [7] Rgraphviz_2.30.0         tools_3.6.1
##  [9] backports_1.1.5          R6_2.4.1
## [11] DBI_1.0.0                lazyeval_0.2.2
## [13] colorspace_1.4-1         withr_2.1.2
## [15] tidyselect_0.2.5         prettyunits_1.0.2
## [17] bit_1.1-14               curl_4.2
## [19] compiler_3.6.1           graph_1.64.0
## [21] formatR_1.7              rtracklayer_1.46.0
## [23] bookdown_0.15            checkmate_1.9.4
## [25] scales_1.1.0             genefilter_1.68.0
## [27] RBGL_1.62.1              askpass_1.1
## [29] rappdirs_0.3.1           stringr_1.4.0
## [31] digest_0.6.22            rmarkdown_1.17
## [33] AnnotationForge_1.28.0   pkgconfig_2.0.3
## [35] htmltools_0.4.0          BSgenome_1.54.0
## [37] dbplyr_1.4.2             limma_3.42.0
## [39] rlang_0.4.1              RSQLite_2.1.2
## [41] GOstats_2.52.0           hwriter_1.3.2
## [43] dplyr_0.8.3              VariantAnnotation_1.32.0
## [45] RCurl_1.95-4.12          magrittr_1.5
## [47] GO.db_3.10.0             GenomeInfoDbData_1.2.2
## [49] Matrix_1.2-17            Rcpp_1.0.3
## [51] munsell_0.5.0            lifecycle_0.1.0
## [53] stringi_1.4.3            yaml_2.2.0
## [55] edgeR_3.28.0             zlibbioc_1.32.0
## [57] BiocFileCache_1.10.2     grid_3.6.1
## [59] blob_1.2.0               crayon_1.3.4
## [61] lattice_0.20-38          splines_3.6.1
## [63] GenomicFeatures_1.38.0   annotate_1.64.0
## [65] hms_0.5.2                batchtools_0.9.11
## [67] locfit_1.5-9.1           zeallot_0.1.0
## [69] knitr_1.26               pillar_1.4.2
## [71] rjson_0.2.20             base64url_1.4
## [73] codetools_0.2-16         biomaRt_2.42.0
## [75] XML_3.98-1.20            glue_1.3.1
## [77] evaluate_0.14            latticeExtra_0.6-28
## [79] data.table_1.12.6        BiocManager_1.30.10
## [81] vctrs_0.2.0              gtable_0.3.0
## [83] openssl_1.4.1            purrr_0.3.3
## [85] assertthat_0.2.1         ggplot2_3.2.1
## [87] xfun_0.11                xtable_1.8-4
## [89] survival_2.44-1.1        pheatmap_1.0.12
## [91] tibble_2.1.3             AnnotationDbi_1.48.0
## [93] memoise_1.1.0            brew_1.0-6
## [95] GSEABase_1.48.0
```

13 Funding

This project was supported by funds from the National Institutes of Health (NIH) and the National Science Foundation (NSF).

References

H Backman, Tyler W, and Thomas Girke. 2016. "systemPipeR: NGS workflow and report generation environment." *BMC Bioinformatics* 17 (1): 388. https://doi.org/10.1186/s12859-016-1241-0.

Li, H, and R Durbin. 2009. "Fast and Accurate Short Read Alignment with Burrows-Wheeler Transform." *Bioinformatics* 25 (14): 1754–60. https://doi.org/10.1093/bioinformatics/btp324.

Li, Heng. 2011. "A Statistical Framework for SNP Calling, Mutation Discovery, Association Mapping and Population Genetical Parameter Estimation from Sequencing Data." *Bioinformatics* 27 (21): 2987–93. https://doi.org/10.1093/bioinformatics/btr509.

———. 2013. "Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM." *arXiv [Q-bio.GN]*, March. http://arxiv.org/abs/1303.3997.

McKenna, Aaron, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernytsky, Kiran Garimella, et al. 2010. "The Genome Analysis Toolkit: A MapReduce Framework for Analyzing Next-Generation DNA Sequencing Data." *Genome Res.* 20 (9): 1297–1303. https://doi.org/10.1101/gr.107524.110.