systemPipeR: NGS workflow and report generation environment

Author: Thomas Girke (thomas.girke@ucr.edu)

Last update: 22 June, 2016

Contents

1 Introduction Slides **1**

2 Background **1**

3 Getting Started **2**
3.1 Download latest version of this tutorial . . . 3
3.2 Installation . . . 3
3.3 Loading package and documentation . . . 3
3.4 Load sample data and workflow templates . . . 3
3.5 Structure of *targets* file . . . 4
3.5.1 Structure of *targets* file for single end (SE) samples . . . 4
3.5.2 Structure of *targets* file for paired end (PE) samples . . . 5
3.5.3 Sample comparisons . . . 5
3.6 Structure of *param* file and *SYSargs* container . . . 5

4 More detail **6**

5 Workflow demo **6**

6 Version information **7**

References **7**

Note: the most recent version of this tutorial can be found here and a short overview slide show here.

1 Introduction Slides

See here.

2 Background

systemPipeR provides utilities for building and running automated end-to-end analysis workflows for a wide range of next generation sequence (NGS) applications such as RNA-Seq, ChIP-Seq, VAR-Seq and Ribo-Seq (Girke 2014). Important features include a uniform workflow interface across different NGS applications, automated report generation, and support for running both R and command-line software, such as NGS aligners or peak/variant callers, on local computers or compute clusters. The latter supports interactive job submissions and batch submissions to queuing systems of clusters. For instance, *systemPipeR* can be used with most command-line aligners such as `BWA` (Heng Li 2013; H Li and Durbin 2009), `TopHat2` (Kim et al. 2013) and `Bowtie2` (Langmead and Salzberg 2012), as well as the R-based NGS aligners *Rsubread* (Liao, Smyth, and Shi 2013) and *gsnap (gmapR)* (Wu and Nacu 2010). Efficient handling of complex sample sets (*e.g.* FASTQ/BAM files) and experimental designs is facilitated by a well-defined sample annotation infrastructure

which improves reproducibility and user-friendliness of many typical analysis workflows in the NGS area (Lawrence et al. 2013).

Motivation and advantages of *sytemPipeR* environment:

1. Facilitates design of complex NGS workflows involving multiple R/Bioconductor packages
2. Common workflow interface for different NGS applications
3. Makes NGS analysis with Bioconductor utilities more accessible to new users
4. Simplifies usage of command-line software from within R
5. Reduces complexity of using compute clusters for R and command-line software
6. Accelerates runtime of workflows via parallelzation on computer systems with mutiple CPU cores and/or multiple compute nodes
7. Automates generation of analysis reports to improve reproducibility

A central concept for designing workflows within the *sytemPipeR* environment is the use of workflow management containers called *SYSargs* (see Figure 1). Instances of this S4 object class are constructed by the *systemArgs* function from two simple tabular files: a *targets* file and a *param* file. The latter is optional for workflow steps lacking command-line software. Typically, a *SYSargs* instance stores all sample-level inputs as well as the paths to the corresponding outputs generated by command-line- or R-based software generating sample-level output files, such as read preprocessors (trimmed/filtered FASTQ files), aligners (SAM/BAM files), variant callers (VCF/BCF files) or peak callers (BED/WIG files). Each sample level input/outfile operation uses its own *SYSargs* instance. The outpaths of *SYSargs* usually define the sample inputs for the next *SYSargs* instance. This connectivity is established by writing the outpaths with the *writeTargetsout* function to a new *targets* file that serves as input to the next *systemArgs* call. Typically, the user has to provide only the initial *targets* file. All downstream *targets* files are generated automatically. By chaining several *SYSargs* steps together one can construct complex workflows involving many sample-level input/output file operations with any combinaton of command-line or R-based software.

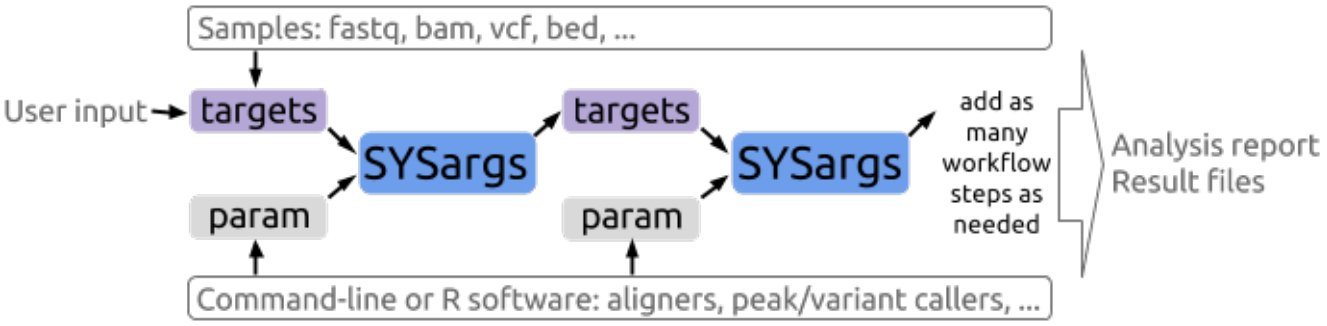

Figure 1: Workflow design structure of *systemPipeR*

The intended way of running *sytemPipeR* workflows is via **.Rnw* or **.Rmd* files, which can be executed either line-wise in interactive mode or with a single command from R or the command-line using a *Makefile*. This way comprehensive and reproducible analysis reports can be generated in PDF or HTML format in a fully automated manner by making use of the highly functional reporting utilities available for R. Templates for setting up custom project reports are provided as **.Rnw* files by the helper package *systemPipeRdata* and in the vignettes subdirectory of *systemPipeR*. The corresponding PDFs of these report templates are available here: *systemPipeRNAseq*, *systemPipeRIBOseq*, *systemPipeChIPseq* and *systemPipeVARseq*. To work with **.Rnw* or **.Rmd* files efficiently, basic knowledge of *Sweave* or *knitr* and *Latex* or *R Markdown v2* is required.

Back to Table of Contents

3 Getting Started

3.1 Download latest version of this tutorial

In case there is a newer version of this tutorial, download its `systemPipeR_Intro.Rmd` source and open it in your R IDE (e.g. vim-r or RStudio).

```
download.file("https://raw.githubusercontent.com/tgirke/systemPipeRdata/master/vignettes/systemPipeR_Intro
```

3.2 Installation

The R software for running *systemPipeR* can be downloaded from *CRAN*. The *systemPipeR* environment can be installed from the R console using the *biocLite* install command. The associated data package *systemPipeRdata* can be installed the same way. The latter is a helper package for generating *systemPipeR* workflow environments with a single command containing all parameter files and sample data required to quickly test and run workflows.

```
source("http://bioconductor.org/biocLite.R") # Sources the biocLite.R installation script
biocLite("systemPipeR") # Installs systemPipeR
biocLite("systemPipeRdata") # Installs systemPipeRdata
```

Back to Table of Contents

3.3 Loading package and documentation

```
library("systemPipeR") # Loads the package
library(help="systemPipeR") # Lists package info
vignette("systemPipeR") # Opens vignette
```

Back to Table of Contents

3.4 Load sample data and workflow templates

The mini sample FASTQ files used by this overview vignette as well as the associated workflow reporting vignettes can be loaded via the *systemPipeRdata* package as shown below. The chosen data set SRP010938 contains 18 paired-end (PE) read sets from *Arabidposis thaliana* (Howard et al. 2013). To minimize processing time during testing, each FASTQ file has been subsetted to 90,000-100,000 randomly sampled PE reads that map to the first 100,000 nucleotides of each chromosome of the *A. thalina* genome. The corresponding reference genome sequence (FASTA) and its GFF annotion files (provided in the same download) have been truncated accordingly. This way the entire test sample data set requires less than 200MB disk storage space. A PE read set has been chosen for this test data set for flexibility, because it can be used for testing both types of analysis routines requiring either SE (single end) reads or PE reads.

The following generates a fully populated *systemPipeR* workflow environment (here for RNA-Seq) in the current working directory of an R session. At this time the package includes workflow templates for RNA-Seq, ChIP-Seq, VAR-Seq and Ribo-Seq. Templates for additional NGS applications will be provided in the future.

```
library(systemPipeRdata)
genWorkenvir(workflow="riboseq", bam=TRUE)
setwd("riboseq")
```

The working environment of the sample data loaded in the previous step contains the following preconfigured directory structure. Directory names are indicated in ***grey***. Users can change this structure as needed, but need to adjust the code in their workflows accordingly.

- ***workflow/*** (*e.g. rnaseq/*)
  - This is the directory of the R session running the workflow.
  - Run script (**.Rnw* or **.Rmd*) and sample annotation (*targets.txt*) files are located here.

  - Note, this directory can have any name (*e.g.* ***rnaseq***, ***varseq***). Changing its name does not require any modifications in the run script(s).
  - Important subdirectories:
    * ***param/***
      · Stores parameter files such as: **.param*, **.tmpl* and **_run.sh*.
    * ***data/***
      · FASTQ samples
      · Reference FASTA file
      · Annotations
      · etc.
    * ***results/***
      · Alignment, variant and peak files (BAM, VCF, BED)
      · Tabular result files
      · Images and plots
      · etc.

The following parameter files are included in each workflow template:

1. *targets.txt*: initial one provided by user; downstream *targets_*.txt* files are generated automatically
2. **.param*: defines parameter for input/output file operations, *e.g.* *trim.param*, *bwa.param*, *vartools.parm*, ...
3. **_run.sh*: optional bash script, *e.g.*: *gatk_run.sh*
4. Compute cluster environment (skip on single machine):
   - *.BatchJobs*: defines type of scheduler for *BatchJobs*
   - **.tmpl*: specifies parameters of scheduler used by a system, *e.g.* Torque, SGE, StarCluster, Slurm, etc.

Back to Table of Contents

3.5 Structure of *targets* file

The *targets* file defines all input files (*e.g.* FASTQ, BAM, BCF) and sample comparisons of an analysis workflow. The following shows the format of a sample *targets* file included in the package. It also can be viewed and downloaded from *systemPipeR*'s GitHub repository here. In a target file with a single type of input files, here FASTQ files of single end (SE) reads, the first three columns are mandatory including their column names, while it is four mandatory columns for FASTQ files of PE reads. All subsequent columns are optional and any number of additional columns can be added as needed.

3.5.1 Structure of *targets* file for single end (SE) samples

```
library(systemPipeR)
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
read.delim(targetspath, comment.char = "#")
##                    FileName SampleName Factor SampleLong Experiment        Date
## 1  ./data/SRR446027_1.fastq        M1A     M1  Mock.1h.A          1 23-Mar-2012
## 2  ./data/SRR446028_1.fastq        M1B     M1  Mock.1h.B          1 23-Mar-2012
## 3  ./data/SRR446029_1.fastq        A1A     A1   Avr.1h.A          1 23-Mar-2012
## 4  ./data/SRR446030_1.fastq        A1B     A1   Avr.1h.B          1 23-Mar-2012
## 5  ./data/SRR446031_1.fastq        V1A     V1   Vir.1h.A          1 23-Mar-2012
## 6  ./data/SRR446032_1.fastq        V1B     V1   Vir.1h.B          1 23-Mar-2012
## 7  ./data/SRR446033_1.fastq        M6A     M6  Mock.6h.A          1 23-Mar-2012
## 8  ./data/SRR446034_1.fastq        M6B     M6  Mock.6h.B          1 23-Mar-2012
## 9  ./data/SRR446035_1.fastq        A6A     A6   Avr.6h.A          1 23-Mar-2012
## 10 ./data/SRR446036_1.fastq        A6B     A6   Avr.6h.B          1 23-Mar-2012
## 11 ./data/SRR446037_1.fastq        V6A     V6   Vir.6h.A          1 23-Mar-2012
```

```
## 12 ./data/SRR446038_1.fastq        V6B     V6   Vir.6h.B          1 23-Mar-2012
## 13 ./data/SRR446039_1.fastq       M12A    M12 Mock.12h.A          1 23-Mar-2012
## 14 ./data/SRR446040_1.fastq       M12B    M12 Mock.12h.B          1 23-Mar-2012
## 15 ./data/SRR446041_1.fastq       A12A    A12  Avr.12h.A          1 23-Mar-2012
## 16 ./data/SRR446042_1.fastq       A12B    A12  Avr.12h.B          1 23-Mar-2012
## 17 ./data/SRR446043_1.fastq       V12A    V12  Vir.12h.A          1 23-Mar-2012
## 18 ./data/SRR446044_1.fastq       V12B    V12  Vir.12h.B          1 23-Mar-2012
```

To work with custom data, users need to generate a *targets* file containing the paths to their own FASTQ files and then provide under *targetspath* the path to the corresponding *targets* file.

Back to Table of Contents

3.5.2 Structure of *targets* file for paired end (PE) samples

```
targetspath <- system.file("extdata", "targetsPE.txt", package="systemPipeR")
read.delim(targetspath, comment.char = "#")[1:2,1:6]
##                  FileName1                FileName2 SampleName Factor SampleLong Experiment
## 1 ./data/SRR446027_1.fastq ./data/SRR446027_2.fastq        M1A     M1  Mock.1h.A          1
## 2 ./data/SRR446028_1.fastq ./data/SRR446028_2.fastq        M1B     M1  Mock.1h.B          1
```

Back to Table of Contents

3.5.3 Sample comparisons

Sample comparisons are defined in the header lines of the *targets* file starting with '# <CMP>'.

```
readLines(targetspath)[1:4]
## [1] "# Project ID: Arabidopsis - Pseudomonas alternative splicing study (SRA: SRP010938; PMID: 24098335
## [2] "# The following line(s) allow to specify the contrasts needed for comparative analyses, such as DE
## [3] "# <CMP> CMPset1: M1-A1, M1-V1, A1-V1, M6-A6, M6-V6, A6-V6, M12-A12, M12-V12, A12-V12"
## [4] "# <CMP> CMPset2: ALL"
```

The function *readComp* imports the comparison information and stores it in a *list*. Alternatively, *readComp* can obtain the comparison information from the corresponding *SYSargs* object (see below). Note, these header lines are optional. They are mainly useful for controlling comparative analyses according to certain biological expectations, such as identifying differentially expressed genes in RNA-Seq experiments based on simple pair-wise comparisons.

```
readComp(file=targetspath, format="vector", delim="-")
## $CMPset1
## [1] "M1-A1"   "M1-V1"   "A1-V1"   "M6-A6"   "M6-V6"   "A6-V6"   "M12-A12" "M12-V12" "A12-V12"
##
## $CMPset2
##  [1] "M1-A1"   "M1-V1"   "M1-M6"   "M1-A6"   "M1-V6"   "M1-M12"  "M1-A12"  "M1-V12"  "A1-V1"
## [10] "A1-M6"   "A1-A6"   "A1-V6"   "A1-M12"  "A1-A12"  "A1-V12"  "V1-M6"   "V1-A6"   "V1-V6"
## [19] "V1-M12"  "V1-A12"  "V1-V12"  "M6-A6"   "M6-V6"   "M6-M12"  "M6-A12"  "M6-V12"  "A6-V6"
## [28] "A6-M12"  "A6-A12"  "A6-V12"  "V6-M12"  "V6-A12"  "V6-V12"  "M12-A12" "M12-V12" "A12-V12"
```

Back to Table of Contents

3.6 Structure of *param* file and *SYSargs* container

The *param* file defines the parameters of a chosen command-line software. The following shows the format of a sample *param* file provided by this package.

```
parampath <- system.file("extdata", "tophat.param", package="systemPipeR")
```

The *systemArgs* function imports the definitions of both the *param* file and the *targets* file, and stores all relevant information in a *SYSargs* object (S4 class). To run the pipeline without command-line software, one can assign *NULL* to *sysma* instead of a *param* file. In addition, one can start *systemPipeR* workflows with pre-generated BAM files by providing a targets file where the *FileName* column provides the paths to the BAM files. Note, in the following example the usage of *suppressWarnings()* is only relevant for building this vignette. In typical workflows it should be removed.

```
args <- suppressWarnings(systemArgs(sysma=parampath, mytargets=targetspath))
args
## An instance of 'SYSargs' for running 'tophat' on 18 samples
```

Several accessor methods are available that are named after the slot names of the *SYSargs* object.

```
names(args)
##  [1] "targetsin"     "targetsout"    "targetsheader" "modules"       "software"      "cores"
##  [7] "other"         "reference"     "results"       "infile1"       "infile2"       "outfile1"
## [13] "sysargs"       "outpaths"
```

Of particular interest is the *sysargs()* method. It constructs the system commands for running command-lined software as specified by a given *param* file combined with the paths to the input samples (*e.g.* FASTQ files) provided by a *targets* file. The example below shows the *sysargs()* output for running TopHat2 on the first PE read sample. Evaluating the output of *sysargs()* can be very helpful for designing and debugging *param* files of new command-line software or changing the parameter settings of existing ones.

```
sysargs(args)[1]
##
## "tophat -p 4 -g 1 --segment-length 25 -i 30 -I 3000 -o /home/tgirke/Dropbox/Software/systemPipeRdata/gi
modules(args)
## [1] "bowtie2/2.2.5" "tophat/2.0.14"
cores(args)
## [1] 4
outpaths(args)[1]
##
## "/home/tgirke/Dropbox/Software/systemPipeRdata/github/systemPipeRdata/vignettes/results/SRR446027_1.fas
```

The content of the *param* file can also be returned as JSON object as follows (requires *rjson* package).

```
systemArgs(sysma=parampath, mytargets=targetspath, type="json")
## [1] "{\"modules\":{\"n1\":\"\",\"v2\":\"bowtie2/2.2.5\",\"n1\":\"\",\"v2\":\"tophat/2.0.14\"},\"softwar
```

Back to Table of Contents

4 More detail

See systemPipeR vignette here.

5 Workflow demo

RIBO-Seq example here

Back to Table of Contents

6 Version information

```
sessionInfo()
## R version 3.3.0 (2016-05-03)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    methods   stats     graphics  utils     datasets  grDevices base
##
## other attached packages:
##  [1] systemPipeR_1.6.2          ShortRead_1.30.0           GenomicAlignments_1.8.3
##  [4] SummarizedExperiment_1.2.3 Biobase_2.32.0             BiocParallel_1.6.2
##  [7] Rsamtools_1.24.0           Biostrings_2.40.2          XVector_0.12.0
## [10] GenomicRanges_1.24.2       GenomeInfoDb_1.8.1         IRanges_2.6.1
## [13] S4Vectors_0.10.1           BiocGenerics_0.18.0        BiocStyle_2.0.2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.5             lattice_0.20-33         GO.db_3.3.0             digest_0.6.9
##  [5] plyr_1.8.4              BatchJobs_1.6           backports_1.0.2         RSQLite_1.0.0
##  [9] evaluate_0.9            ggplot2_2.1.0           zlibbioc_1.18.0         GenomicFeatures_1.24.3
## [13] annotate_1.50.0         Matrix_1.2-6            checkmate_1.8.0         rmarkdown_0.9.6
## [17] GOstats_2.38.0          splines_3.3.0           stringr_1.0.0           pheatmap_1.0.8
## [21] RCurl_1.95-4.8          biomaRt_2.28.0          munsell_0.4.3           sendmailR_1.2-1
## [25] rtracklayer_1.32.1      base64enc_0.1-3         BBmisc_1.9              htmltools_0.3.5
## [29] fail_1.3                edgeR_3.14.0            codetools_0.2-14        XML_3.98-1.4
## [33] AnnotationForge_1.14.2 bitops_1.0-6            grid_3.3.0              RBGL_1.48.1
## [37] xtable_1.8-2            GSEABase_1.34.0         gtable_0.2.0            DBI_0.4-1
## [41] magrittr_1.5            formatR_1.4             scales_0.4.0            graph_1.50.0
## [45] stringi_1.1.1           hwriter_1.3.2           genefilter_1.54.2       limma_3.28.10
## [49] latticeExtra_0.6-28     brew_1.0-6              rjson_0.2.15            RColorBrewer_1.1-2
## [53] tools_3.3.0             Category_2.38.0         survival_2.39-4         yaml_2.1.13
## [57] AnnotationDbi_1.34.3    colorspace_1.2-6        knitr_1.13
```

Back to Table of Contents

References

Girke, Thomas. 2014. "systemPipeR: NGS Workflow and Report Generation Environment." UC Riverside. https://github.com/tgirke/systemPipeR.

Howard, Brian E, Qiwen Hu, Ahmet Can Babaoglu, Manan Chandra, Monica Borghi, Xiaoping Tan, Luyan He, et al. 2013. "High-Throughput RNA Sequencing of Pseudomonas-Infected Arabidopsis Reveals Hidden Transcriptome Complexity and Novel Splice Variants." *PLoS One* 8 (10): e74183. doi:10.1371/journal.pone.0074183.

Kim, Daehwan, Geo Pertea, Cole Trapnell, Harold Pimentel, Ryan Kelley, and Steven L Salzberg. 2013. "TopHat2: Accurate Alignment of Transcriptomes in the Presence of Insertions, Deletions and Gene Fusions." *Genome Biol.* 14 (4):

R36. doi:10.1186/gb-2013-14-4-r36.

Langmead, Ben, and Steven L Salzberg. 2012. "Fast Gapped-Read Alignment with Bowtie 2." *Nat. Methods* 9 (4). Nature Publishing Group: 357–59. doi:10.1038/nmeth.1923.

Lawrence, Michael, Wolfgang Huber, Hervé Pagès, Patrick Aboyoun, Marc Carlson, Robert Gentleman, Martin T Morgan, and Vincent J Carey. 2013. "Software for Computing and Annotating Genomic Ranges." *PLoS Comput. Biol.* 9 (8): e1003118. doi:10.1371/journal.pcbi.1003118.

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

Li, Heng. 2013. "Aligning Sequence Reads, Clone Sequences and Assembly Contigs with BWA-MEM." *ArXiv [Q-Bio.GN]*. http://arxiv.org/abs/1303.3997.

Liao, Yang, Gordon K Smyth, and Wei Shi. 2013. "The Subread Aligner: Fast, Accurate and Scalable Read Mapping by Seed-and-Vote." *Nucleic Acids Res.* 41 (10): e108. doi:10.1093/nar/gkt214.

Wu, T D, and S Nacu. 2010. "Fast and SNP-tolerant Detection of Complex Variants and Splicing in Short Reads." *Bioinformatics* 26 (7): 873–81. doi:10.1093/bioinformatics/btq057.