# RNA-Seq Workflow Template

*First/last name (first.last@ucr.edu)*

*Last update: 05 June, 2017*

## Introduction

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

## Samples and environment settings

### Environment settings and input data

Typically, the user wants to record here the sources and versions of the reference genome sequence along with the corresponding annotations. In the provided sample data set all data inputs are stored in a `data` subdirectory and all results will be written to a separate `results` directory, while the `systemPipeRNAseq.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.

To run this sample report, mini sample FASTQ and reference genome files can be downloaded from here. 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 is less than 200MB in 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 loads one of the available NGS workflow templates (here RNA-Seq) into the user's current working directory. At the moment, 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="rnaseq")
setwd("rnaseq")
```

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

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

Now open the R markdown script `systemPipeRNAseq.Rmd`in your R IDE (*e.g.* vim-r or RStudio) and run the workflow as outlined below. If you work under Vim-R-Tmux, the following command sequence will connect the user in an interactive session with a node on the cluster. The code of the `Rmd` script can then be sent from Vim on the login (head) node to an open R session running on the corresponding computer node. This is important since Tmux sessions should not be run on the computer nodes.

```
# push `F2` on your keyboard to open interactive R session
q("no") # closes R session on head node
srun --x11 --partition=intel --mem=2gb --cpus-per-task 1 --ntasks 1 --time 2:00:00 --pty bash -l
R
```

Now check whether your R session is running on a computer node of the cluster and not on a head node.

```
system("hostname") # should return name of a compute node starting with i or c
getwd() # checks current working directory of R session
dir() # returns content of current working directory
```

### Required packages and resources

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

```
library(systemPipeR)
```

If applicable load custom functions not provided by

```
source("systemPipeRNAseq_Fct.R")
```

### Experiment definition provided by `targets` file

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

```
targetspath <- system.file("extdata", "targets.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")[,1:4]
targets
```

```
##                    FileName SampleName Factor SampleLong
## 1  ./data/SRR446027_1.fastq        M1A     M1  Mock.1h.A
## 2  ./data/SRR446028_1.fastq        M1B     M1  Mock.1h.B
## 3  ./data/SRR446029_1.fastq        A1A     A1   Avr.1h.A
## 4  ./data/SRR446030_1.fastq        A1B     A1   Avr.1h.B
## 5  ./data/SRR446031_1.fastq        V1A     V1   Vir.1h.A
## 6  ./data/SRR446032_1.fastq        V1B     V1   Vir.1h.B
## 7  ./data/SRR446033_1.fastq        M6A     M6  Mock.6h.A
## 8  ./data/SRR446034_1.fastq        M6B     M6  Mock.6h.B
## 9  ./data/SRR446035_1.fastq        A6A     A6   Avr.6h.A
## 10 ./data/SRR446036_1.fastq        A6B     A6   Avr.6h.B
## 11 ./data/SRR446037_1.fastq        V6A     V6   Vir.6h.A
## 12 ./data/SRR446038_1.fastq        V6B     V6   Vir.6h.B
## 13 ./data/SRR446039_1.fastq       M12A    M12 Mock.12h.A
## 14 ./data/SRR446040_1.fastq       M12B    M12 Mock.12h.B
## 15 ./data/SRR446041_1.fastq       A12A    A12  Avr.12h.A
## 16 ./data/SRR446042_1.fastq       A12B    A12  Avr.12h.B
## 17 ./data/SRR446043_1.fastq       V12A    V12  Vir.12h.A
## 18 ./data/SRR446044_1.fastq       V12B    V12  Vir.12h.B
```

## Read preprocessing

### Read quality filtering and trimming

The function `preprocessReads` allows to apply predefined or custom read preprocessing functions to all FASTQ files referenced in a `SYSargs` container, such as quality filtering or adaptor trimming routines. The

following example performs adaptor trimming with the `trimLRPatterns` function from the `Biostrings` package. After the trimming step a new targets file is generated (here `targets_trim.txt`) containing the paths to the trimmed FASTQ files. The new targets file can be used for the next workflow step with an updated `SYSargs` instance, *e.g.* running the NGS alignments using the trimmed FASTQ files.

```
args <- systemArgs(sysma="param/trim.param", mytargets="targets.txt")
preprocessReads(args=args, Fct="trimLRPatterns(Rpattern='GCCCGGGTAA', subject=fq)",
                batchsize=100000, overwrite=TRUE, compress=TRUE)
writeTargetsout(x=args, file="targets_trim.txt", overwrite=TRUE)
```

### 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`.

```
args <- systemArgs(sysma="param/tophat.param", mytargets="targets.txt")
fqlist <- seeFastq(fastq=infile1(args), batchsize=100000, klength=8)
pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

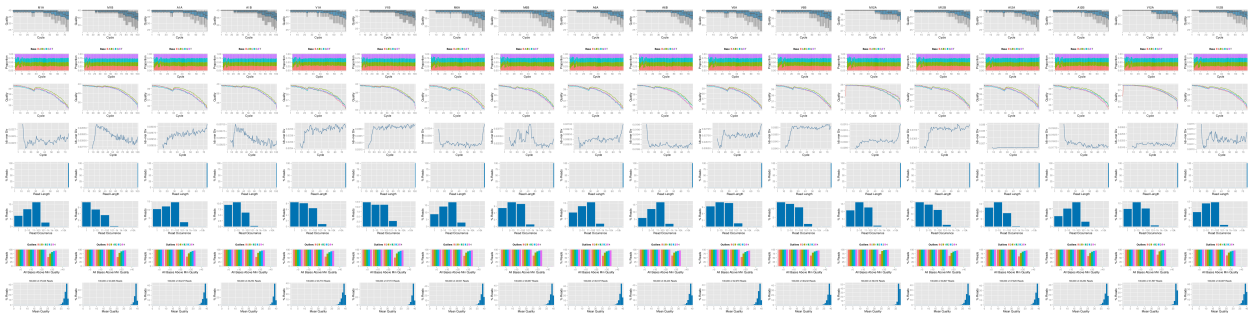

Figure 1: FASTQ quality report for 18 samples

## Alignments

### Read mapping with `Bowtie2/Tophat2`

The NGS reads of this project will be aligned against the reference genome sequence using `Bowtie2/TopHat2` (Kim et al. 2013; Langmead and Salzberg 2012). The parameter settings of the aligner are defined in the `tophat.param` file.

```
args <- systemArgs(sysma="param/tophat.param", mytargets="targets.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
```

Submission of alignment jobs to compute cluster, here using 72 CPU cores (18 `qsub` processes each with 4 CPU cores).

```
moduleload(modules(args))
system("bowtie2-build ./data/tair10.fasta ./data/tair10.fasta")
resources <- list(walltime="20:00:00", ntasks=1, ncpus=cores(args), memory="10G")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="slurm.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
```

### Read mapping with HISAT2

```
args <- systemArgs(sysma="param/hisat2.param", mytargets="targets.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
moduleload(modules(args))
system("hisat2-build ./data/tair10.fasta ./data/tair10.fasta")
resources <- list(walltime="20:00:00", ntasks=1, ncpus=cores(args), memory="10G")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="slurm.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
```

Check whether all BAM files have been created

```
file.exists(outpaths(args))
```

### Read and alignment stats

The following provides an overview 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")
```

The following shows the alignment statistics for a sample file provided by the systemPipeR package.

```
read.table(system.file("extdata", "alignStats.xls", package="systemPipeR"), header=TRUE)[1:4,]
```

```
##   FileName Nreads2x Nalign Perc_Aligned Nalign_Primary Perc_Aligned_Primary
## 1      M1A   192918 177961     92.24697         177961             92.24697
## 2      M1B   197484 159378     80.70426         159378             80.70426
## 3      A1A   189870 176055     92.72397         176055             92.72397
## 4      A1B   188854 147768     78.24457         147768             78.24457
```

### 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 in the results directory.

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

# Read quantification

## Read counting with `summarizeOverlaps` in parallel mode using multiple cores

Reads overlapping with annotation ranges of interest are counted for each sample using the `summarizeOverlaps` function (Lawrence et al. 2013). The read counting is preformed for exonic gene regions in a non-strand-specific manner while ignoring overlaps among different genes. Subsequently, the expression count values are normalized by *reads per kp per million mapped reads* (RPKM). The raw read count table (`countDFeByg.xls`) and the correspoding RPKM table (`rpkmDFeByg.xls`) are written to separate files in the directory of this project. Parallelization is achieved with the `BiocParallel` package, here using 8 CPU cores.

```
library("GenomicFeatures"); library(BiocParallel)
txdb <- makeTxDbFromGFF(file="data/tair10.gff", format="gff", dataSource="TAIR", organism="Arabidopsis
saveDb(txdb, file="./data/tair10.sqlite")
txdb <- loadDb("./data/tair10.sqlite")
(align <- readGAlignments(outpaths(args)[1])) # Demonstrates how to read bam file into R
eByg <- exonsBy(txdb, by=c("gene"))
bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
multicoreParam <- MulticoreParam(workers=2); register(multicoreParam); registered()
counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg, x, mode="Union",
                                                         ignore.strand=TRUE,
                                                         inter.feature=FALSE,
                                                         singleEnd=TRUE))
countDFeByg <- sapply(seq(along=counteByg), function(x) assays(counteByg[[x]])$counts)
rownames(countDFeByg) <- names(rowRanges(counteByg[[1]])); colnames(countDFeByg) <- names(bfl)
rpkmDFeByg <- apply(countDFeByg, 2, function(x) returnRPKM(counts=x, ranges=eByg))
write.table(countDFeByg, "results/countDFeByg.xls", col.names=NA, quote=FALSE, sep="\t")
write.table(rpkmDFeByg, "results/rpkmDFeByg.xls", col.names=NA, quote=FALSE, sep="\t")
```

Sample of data slice of count table

```
read.delim("results/countDFeByg.xls", row.names=1, check.names=FALSE)[1:4,1:5]
```

Sample of data slice of RPKM table

```
read.delim("results/rpkmDFeByg.xls", row.names=1, check.names=FALSE)[1:4,1:4]
```

Note, for most statistical differential expression or abundance analysis methods, such as `edgeR` or `DESeq2`, the raw count values should be used as input. The usage of RPKM values should be restricted to specialty applications required by some users, *e.g.* manually comparing the expression levels among different genes or features.

## Sample-wise correlation analysis

The following computes the sample-wise Spearman correlation coefficients from the `rlog` transformed expression values generated with the `DESeq2` package. After transformation to a distance matrix, hierarchical

clustering is performed with the `hclust` function and the result is plotted as a dendrogram (also see file `sample_tree.pdf`).

```
library(DESeq2, quietly=TRUE); library(ape,  warn.conflicts=FALSE)
countDF <- as.matrix(read.table("./results/countDFeByg.xls"))
colData <- data.frame(row.names=targetsin(args)$SampleName, condition=targetsin(args)$Factor)
dds <- DESeqDataSetFromMatrix(countData = countDF, colData = colData, design = ~ condition)
d <- cor(assay(rlog(dds)), method="spearman")
hc <- hclust(dist(1-d))
pdf("results/sample_tree.pdf")
plot.phylo(as.phylo(hc), type="p", edge.col="blue", edge.width=2, show.node.label=TRUE, no.margin=TRUE)
dev.off()
```

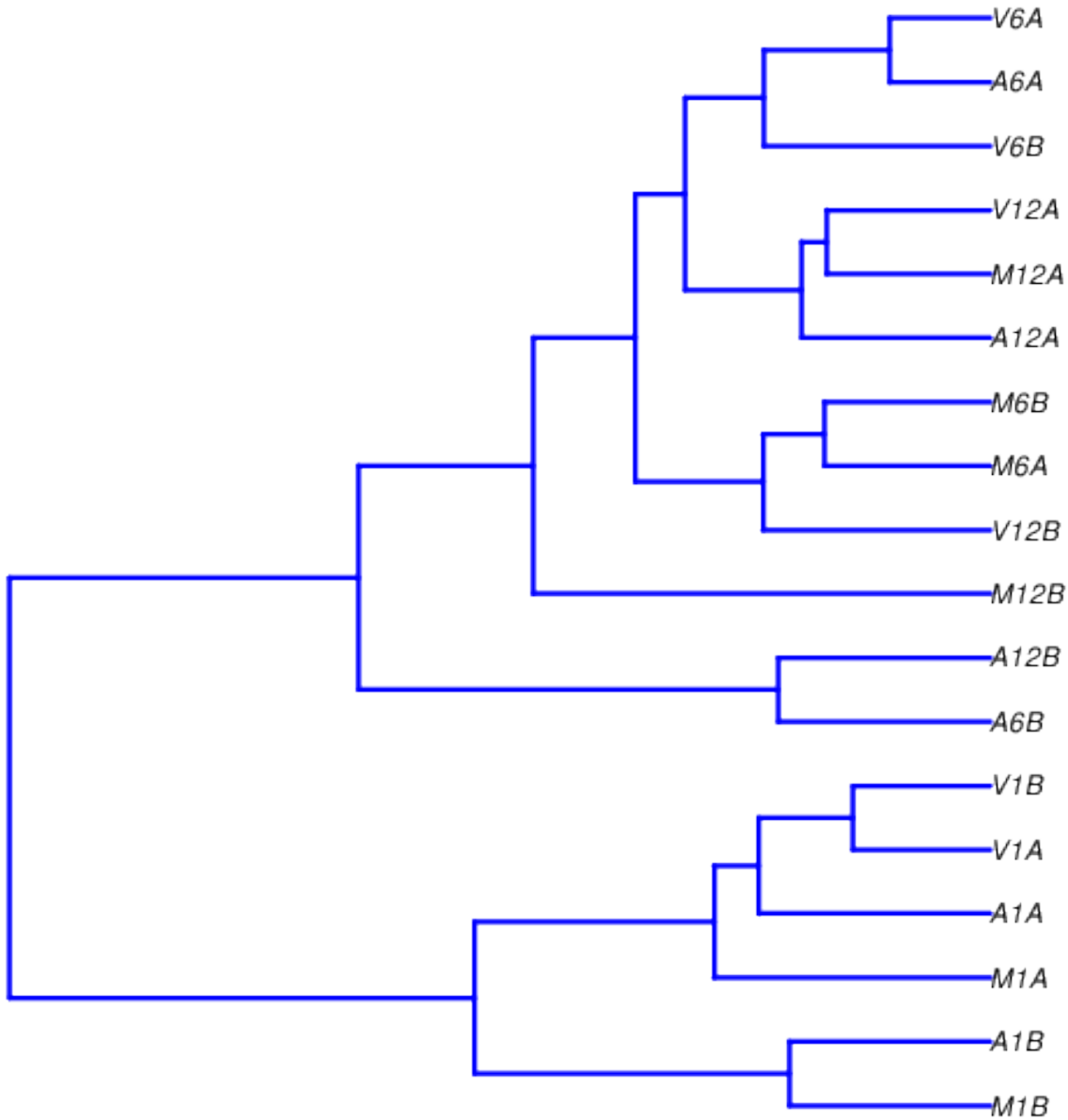


Figure 2: Correlation dendrogram of samples

# Analysis of DEGs

The analysis of differentially expressed genes (DEGs) is performed with the glm method of the `edgeR` package (Robinson, McCarthy, and Smyth 2010). The sample comparisons used by this analysis are defined in the header lines of the `targets.txt` file starting with `<CMP>`.

## Run `edgeR`

```
library(edgeR)
countDF <- read.delim("results/countDFeByg.xls", row.names=1, check.names=FALSE)
targets <- read.delim("targets.txt", comment="#")
cmp <- readComp(file="targets.txt", format="matrix", delim="-")
edgeDF <- run_edgeR(countDF=countDF, targets=targets, cmp=cmp[[1]], independent=FALSE, mdsplot="")
```

Add gene descriptions

```
library("biomaRt")
m <- useMart("plants_mart", dataset="athaliana_eg_gene", host="plants.ensembl.org")
desc <- getBM(attributes=c("tair_locus", "description"), mart=m)
desc <- desc[!duplicated(desc[,1]),]
descv <- as.character(desc[,2]); names(descv) <- as.character(desc[,1])
edgeDF <- data.frame(edgeDF, Desc=descv[rownames(edgeDF)], check.names=FALSE)
write.table(edgeDF, "./results/edgeRglm_allcomp.xls", quote=FALSE, sep="\t", col.names = NA)
```

## Plot DEG results

Filter and plot DEG results for up and down regulated genes. The definition of *up* and *down* is given in the corresponding help file. To open it, type `?filterDEGs` in the R console.

```
edgeDF <- read.delim("results/edgeRglm_allcomp.xls", row.names=1, check.names=FALSE)
pdf("results/DEGcounts.pdf")
DEG_list <- filterDEGs(degDF=edgeDF, filter=c(Fold=2, FDR=20))
dev.off()
write.table(DEG_list$Summary, "./results/DEGcounts.xls", quote=FALSE, sep="\t", row.names=FALSE)
```

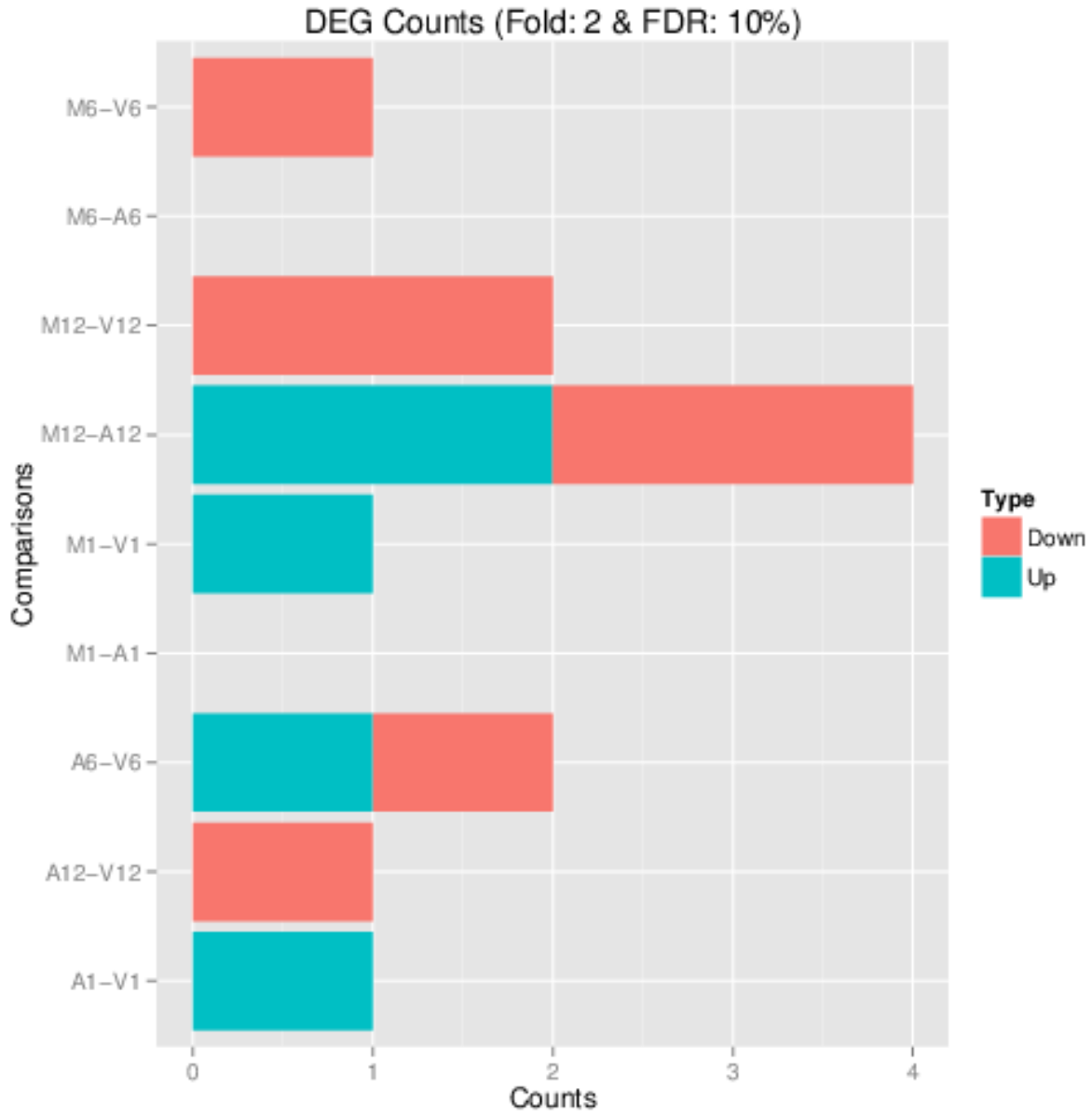


Figure 3: Up and down regulated DEGs with FDR of 1%

## Venn diagrams of DEG sets

The `overLapper` function can compute Venn intersects for large numbers of sample sets (up to 20 or more) and plots 2-5 way Venn diagrams. A useful feature is the possiblity to combine the counts from several Venn comparisons with the same number of sample sets in a single Venn diagram (here for 4 up and down DEG sets).

```
vennsetup <- overLapper(DEG_list$Up[6:9], type="vennsets")
vennsetdown <- overLapper(DEG_list$Down[6:9], type="vennsets")
pdf("results/vennplot.pdf")
vennPlot(list(vennsetup, vennsetdown), mymain="", mysub="", colmode=2, ccol=c("blue", "red"))
```

```
dev.off()
```

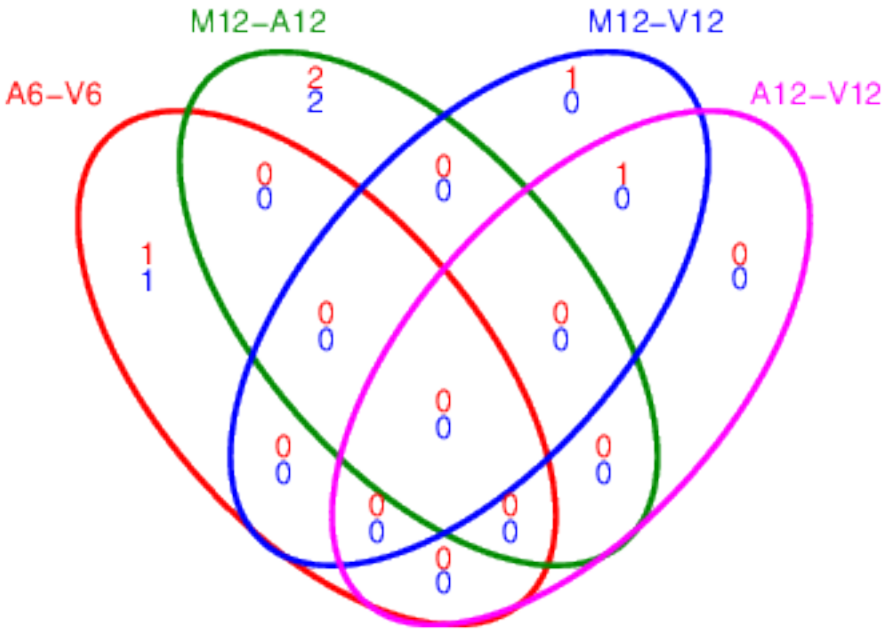


Figure 4: Venn Diagram for 4 Up and Down DEG Sets

# GO term enrichment analysis

## Obtain gene-to-GO mappings

The following shows how to obtain gene-to-GO mappings from `biomaRt` (here for *A. thaliana*) and how to organize them for the downstream GO term enrichment analysis. Alternatively, the gene-to-GO mappings can be obtained for many organisms from Bioconductor's `*.db` genome annotation packages or GO annotation files provided by various genome databases. For each annotation this relatively slow preprocessing step needs to be performed only once. Subsequently, the preprocessed data can be loaded with the `load` function as shown in the next subsection.

```
library("biomaRt")
listMarts() # To choose BioMart database
listMarts(host="plants.ensembl.org")
m <- useMart("plants_mart", host="plants.ensembl.org")
listDatasets(m)
m <- useMart("plants_mart", dataset="athaliana_eg_gene", host="plants.ensembl.org")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_id", "tair_locus", "namespace_1003"), mart=m)
go <- go[go[,3]!="",]; go[,3] <- as.character(go[,3])
go[go[,3]=="molecular_function", 3] <- "F"; go[go[,3]=="biological_process", 3] <- "P"; go[go[,3]=="cel
go[1:4,]
dir.create("./data/GO")
write.table(go, "data/GO/GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, s
catdb <- makeCATdb(myfile="data/GO/GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idc
save(catdb, file="data/GO/catdb.RData")
```

## Batch GO term enrichment analysis

Apply the enrichment analysis to the DEG sets obtained the above differential expression analysis. Note, in the following example the `FDR` filter is set here to an unreasonably high value, simply because of the small size of the toy data set used in this vignette. Batch enrichment analysis of many gene sets is performed with the function. When `method=all`, it returns all GO terms passing the p-value cutoff specified under the `cutoff` arguments. When `method=slim`, it returns only the GO terms specified under the `myslimv` argument. The given example shows how a GO slim vector for a specific organism can be obtained from BioMart.

```
library("biomaRt")
load("data/GO/catdb.RData")
DEG_list <- filterDEGs(degDF=edgeDF, filter=c(Fold=2, FDR=50), plot=FALSE)
up_down <- DEG_list$UporDown; names(up_down) <- paste(names(up_down), "_up_down", sep="")
up <- DEG_list$Up; names(up) <- paste(names(up), "_up", sep="")
down <- DEG_list$Down; names(down) <- paste(names(down), "_down", sep="")
DEGlist <- c(up_down, up, down)
DEGlist <- DEGlist[sapply(DEGlist, length) > 0]
BatchResult <- GOCluster_Report(catdb=catdb, setlist=DEGlist, method="all", id_type="gene", CLSZ=2, cut
library("biomaRt")
m <- useMart("plants_mart", dataset="athaliana_eg_gene", host="plants.ensembl.org")
goslimvec <- as.character(getBM(attributes=c("goslim_goa_accession"), mart=m)[,1])
BatchResultslim <- GOCluster_Report(catdb=catdb, setlist=DEGlist, method="slim", id_type="gene", myslim
```

## Plot batch GO term results

The `data.frame` generated by `GOCluster` can be plotted with the `goBarplot` function. Because of the variable size of the sample sets, it may not always be desirable to show the results from different DEG sets in the same bar plot. Plotting single sample sets is achieved by subsetting the input data frame as shown in the first line of the following example.

```
gos <- BatchResultslim[grep("M6-V6_up_down", BatchResultslim$CLID), ]
gos <- BatchResultslim
pdf("GOslimbarplotMF.pdf", height=8, width=10); goBarplot(gos, gocat="MF"); dev.off()
goBarplot(gos, gocat="BP")
goBarplot(gos, gocat="CC")
```

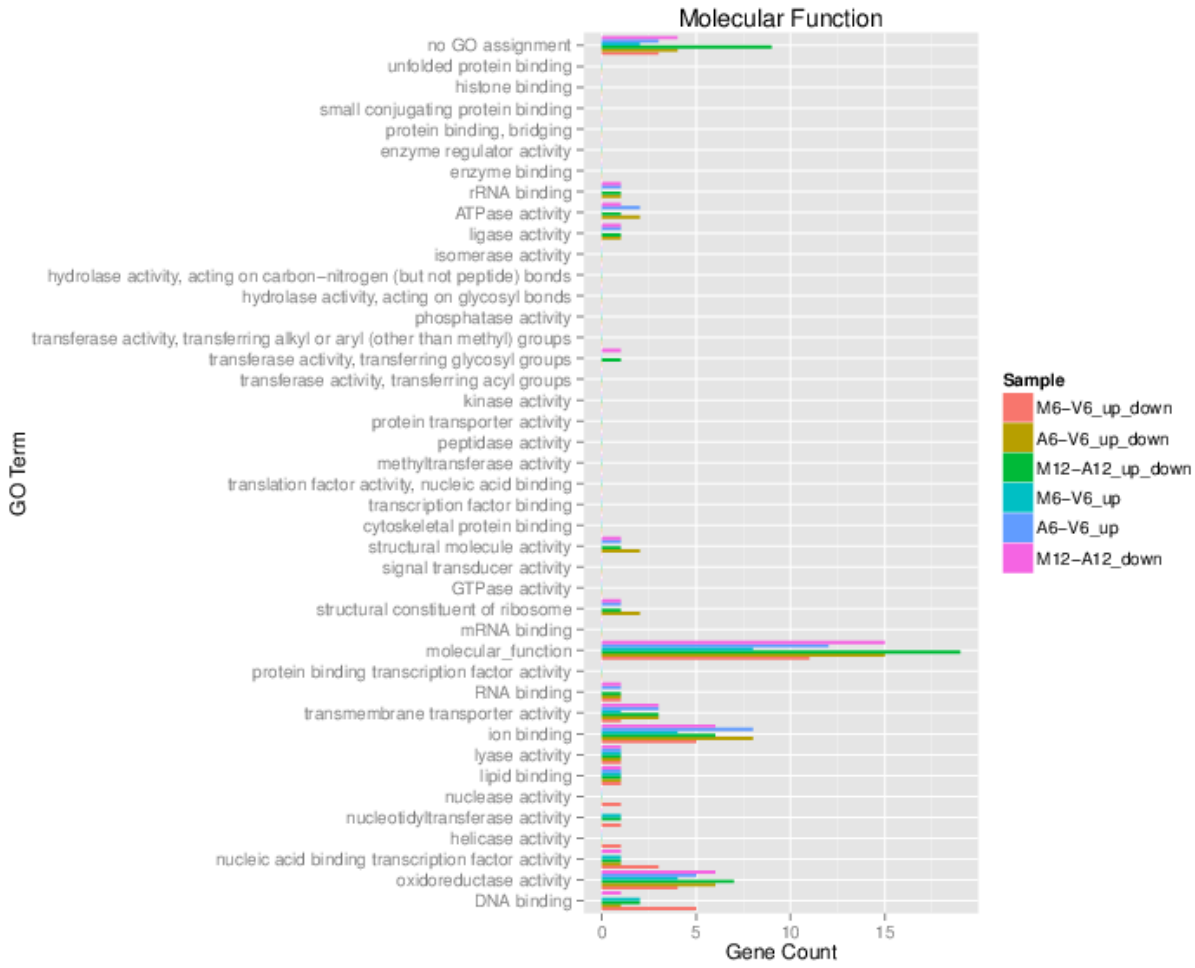


Figure 5: GO Slim Barplot for MF Ontology

## Clustering and heat maps

The following example performs hierarchical clustering on the `rlog` transformed expression matrix subsetted by the DEGs identified in the above differential expression analysis. It uses a Pearson correlation-based distance measure and complete linkage for cluster joining.

```
library(pheatmap)
geneids <- unique(as.character(unlist(DEG_list[[1]])))
y <- assay(rlog(dds))[geneids, ]
pdf("heatmap1.pdf")
pheatmap(y, scale="row", clustering_distance_rows="correlation", clustering_distance_cols="correlation")
dev.off()
```

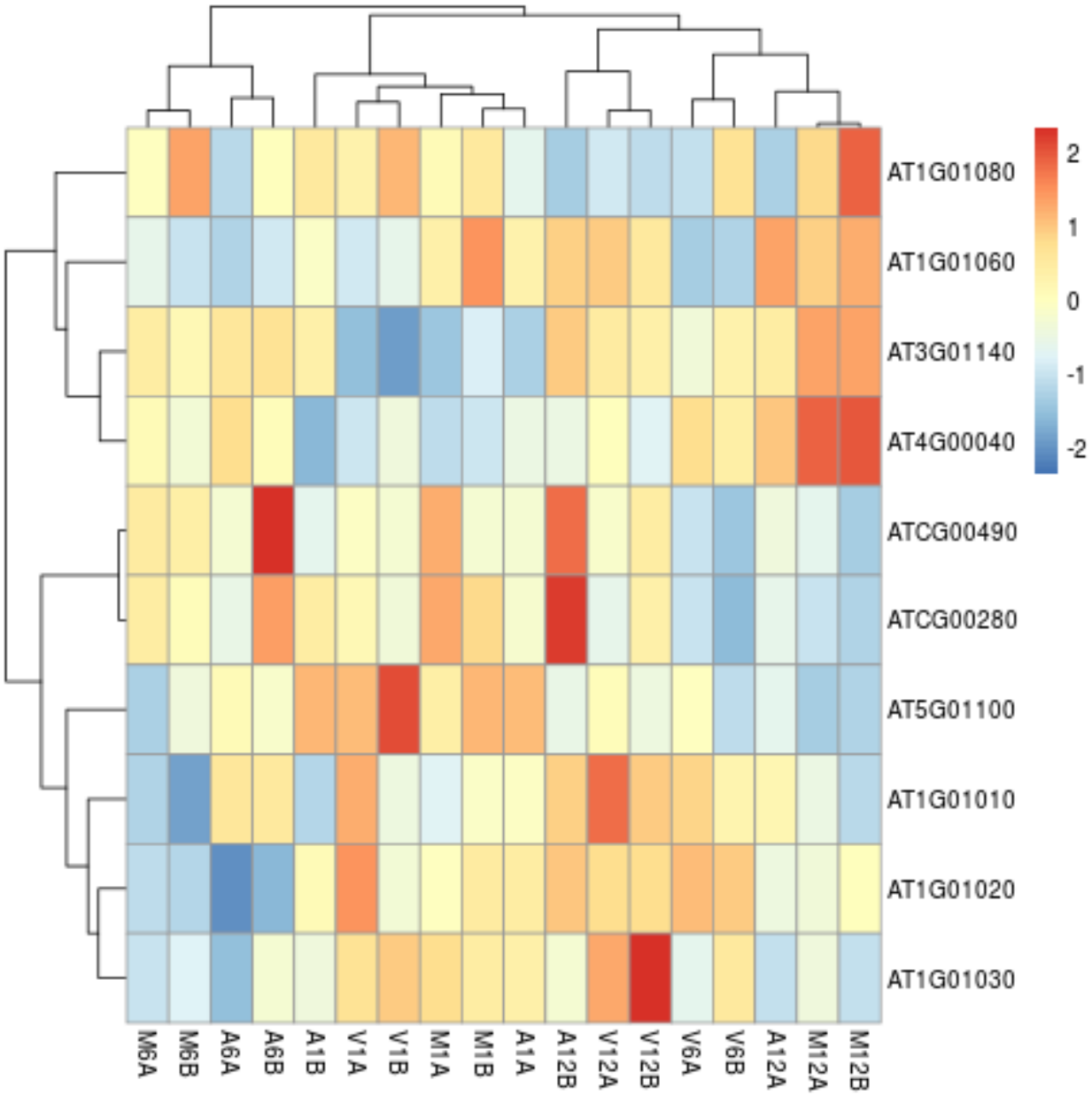


Figure 6: Heat Map with Hierarchical Clustering Dendrograms of DEGs

## Version Information

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.5 LTS
##
## Matrix products: default
```

```
## BLAS: /usr/lib/libblas/libblas.so.3.0
## LAPACK: /usr/lib/lapack/liblapack.so.3.0
##
## 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] stats4    parallel  methods   stats     graphics  utils     datasets  grDevices base
##
## other attached packages:
##  [1] ape_4.1                    ggplot2_2.2.1              systemPipeR_1.10.0
##  [4] ShortRead_1.34.0           GenomicAlignments_1.12.0   SummarizedExperiment_1.6.0
##  [7] DelayedArray_0.2.0         matrixStats_0.52.2         Biobase_2.36.0
## [10] BiocParallel_1.10.0        Rsamtools_1.28.0           Biostrings_2.44.0
## [13] XVector_0.16.0             GenomicRanges_1.28.0       GenomeInfoDb_1.12.0
## [16] IRanges_2.10.0             S4Vectors_0.14.0           BiocGenerics_0.22.0
## [19] BiocStyle_2.4.0
##
## loaded via a namespace (and not attached):
##  [1] edgeR_3.18.0            splines_3.4.0           latticeExtra_0.6-28     RBGL_1.52.0
##  [5] GenomeInfoDbData_0.99.0 yaml_2.1.14             Category_2.42.0         RSQLite_1.1-2
##  [9] backports_1.0.5         lattice_0.20-35         limma_3.32.0            digest_0.6.12
## [13] RColorBrewer_1.1-2      checkmate_1.8.2         colorspace_1.3-2        htmltools_0.3.5
## [17] Matrix_1.2-8            plyr_1.8.4              GSEABase_1.38.0         XML_3.98-1.6
## [21] pheatmap_1.0.8          biomaRt_2.32.0          genefilter_1.58.0       zlibbioc_1.22.0
## [25] xtable_1.8-2            GO.db_3.4.1             scales_0.4.1            brew_1.0-6
## [29] tibble_1.3.0            annotate_1.54.0         GenomicFeatures_1.28.0  lazyeval_0.2.0
## [33] survival_2.41-3         magrittr_1.5            memoise_1.1.0           evaluate_0.10
## [37] fail_1.3                nlme_3.1-131            hwriter_1.3.2           GOstats_2.42.0
## [41] graph_1.54.0            tools_3.4.0             BBmisc_1.11             stringr_1.2.0
## [45] sendmailR_1.2-1         munsell_0.4.3           locfit_1.5-9.1          AnnotationDbi_1.38.0
## [49] compiler_3.4.0          grid_3.4.0              RCurl_1.95-4.8          rjson_0.2.15
## [53] AnnotationForge_1.18.0  bitops_1.0-6            base64enc_0.1-3         rmarkdown_1.5
## [57] codetools_0.2-15        gtable_0.2.0            DBI_0.6-1               knitr_1.15.1
## [61] rtracklayer_1.36.0      rprojroot_1.2           stringi_1.1.5           BatchJobs_1.6
## [65] Rcpp_0.12.10
```

# Funding

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

# References

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

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.

Robinson, M D, D J McCarthy, and G K Smyth. 2010. "EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40. doi:10.1093/bioinformatics/btp616.