

# RNA-Seq workflow template: Some Descriptive Title

Project ID: RNAseq-PI\_Name\_Organism\_Jun2014

Project PI: First Last (first.last@inst.edu)

Author of Report: First Last (first.last@inst.edu)

August 6, 2016

## Contents

---

### 1 Introduction

---

This report describes the analysis of an RNA-Seq project from Dr. First Last's lab which studies the gene expression changes of ... in *organism* .... The experimental design is as follows...

### 2 Sample definitions and environment settings

---

#### 2.1 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.Rnw` 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 *Arabidopsis thaliana* (?). To minimize processing time during testing, each FASTQ file has been subsetting to 90,000-100,000 randomly sampled PE reads that map to the first 100,000 nucleotides of each chromosome of the *A. thaliana* genome. The corresponding reference genome sequence (FASTA) and its GFF annotation 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.

#### 2.2 Required packages and resources

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (?).

```
library(systemPipeR)
```

If applicable load custom functions not provided by `systemPipeR`

```
source("systemPipeRNAseq-Fct.R")
```

## 2.3 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

## 3 Read preprocessing

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

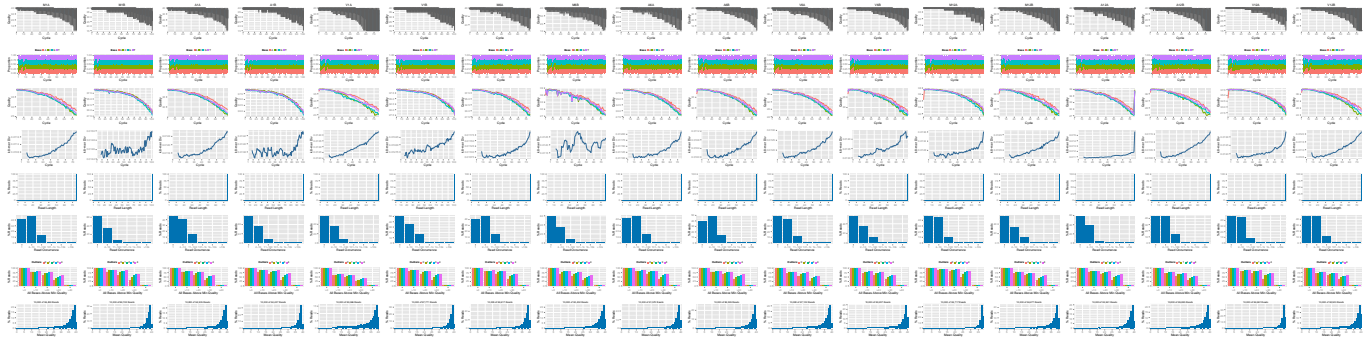


Figure 1: QC report for 18 FASTQ files.

## 4 Alignments

### 4.1 Read mapping with Bowtie2/TopHat2

The NGS reads of this project will be aligned against the reference genome sequence using Bowtie2/TopHat2 (??). The parameter settings of the aligner are defined in the tophat.param file.

```
args <- systemArgs(sysma="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", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
  resourceList=resources)
waitForJobs(reg)
```

Check whether all BAM files have been created

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

### 4.2 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")

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

### 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://biocluster.ucr.edu/~tgirke/",
            urlfile="./results/IGVurl.txt")
```

## 5 Read quantification per annotation range

### 5.1 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 (?). The read counting is performed 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 (`countDfFeByg.xls`) and the corresponding RPKM table (`rpkmDfFeByg.xls`) are written to separate files in the `results` directory of this project. Parallelization is achieved with the *BiocParallel* package, here using 8 CPU cores.

```
library("GenomicFeatures"); library(BiocParallel)
txdb <- loadDb("./data/tair10.sqlite")
eByg <- exonsBy(txdb, by=c("gene"))
bfl <- BamFileList(outpaths(args), yieldSize=50000, index=character())
multicoreParam <- MulticoreParam(workers=8); register(multicoreParam); registered()
counteByg <- bplapply(bfl, function(x) summarizeOverlaps(eByg, x, mode="Union",
                                                         ignore.strand=TRUE,
                                                         inter.feature=FALSE,
                                                         singleEnd=TRUE))

countDfFeByg <- sapply(seq(along=counteByg), function(x) assays(counteByg[[x]])$counts)
rownames(countDfFeByg) <- names(rowRanges(counteByg[[1]])); colnames(countDfFeByg) <- names(bfl)
rpkmDfFeByg <- apply(countDfFeByg, 2, function(x) returnRPKM(counts=x, ranges=eByg))
write.table(countDfFeByg, "results/countDfFeByg.xls", col.names=NA, quote=FALSE, sep="\t")
write.table(rpkmDfFeByg, "results/rpkmDfFeByg.xls", col.names=NA, quote=FALSE, sep="\t")
```

Sample of data slice of count table

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

Sample of data slice of RPKM table

```
read.delim("results/rpkmDfFeByg.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.

### 5.2 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 ([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()
```

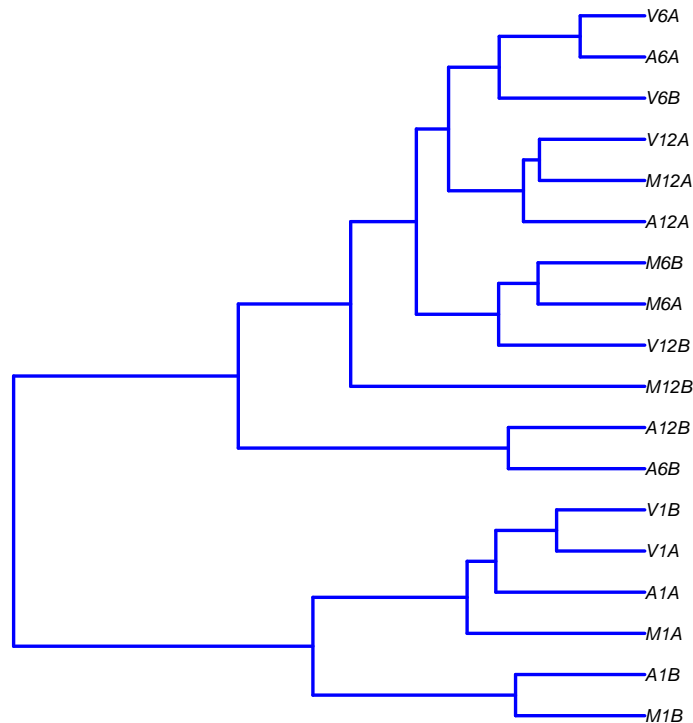


Figure 2: Correlation dendrogram of samples.

## 6 Analysis of differentially expressed genes with *edgeR*

The analysis of differentially expressed genes (DEGs) is performed with the glm method from the *edgeR* package (?). The sample comparisons used by this analysis are defined in the header lines of the `targets` file starting with <CMP>.

```
library(edgeR)
countDF <- read.delim("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 custom functional descriptions. Skip this step if desc.xls is not available.

```
desc <- read.delim("data/desc.xls")
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)
```

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=1))
dev.off()
write.table(DEG_list$Summary, "./results/DEGcounts.xls", quote=FALSE, sep="\t", row.names=FALSE)
```

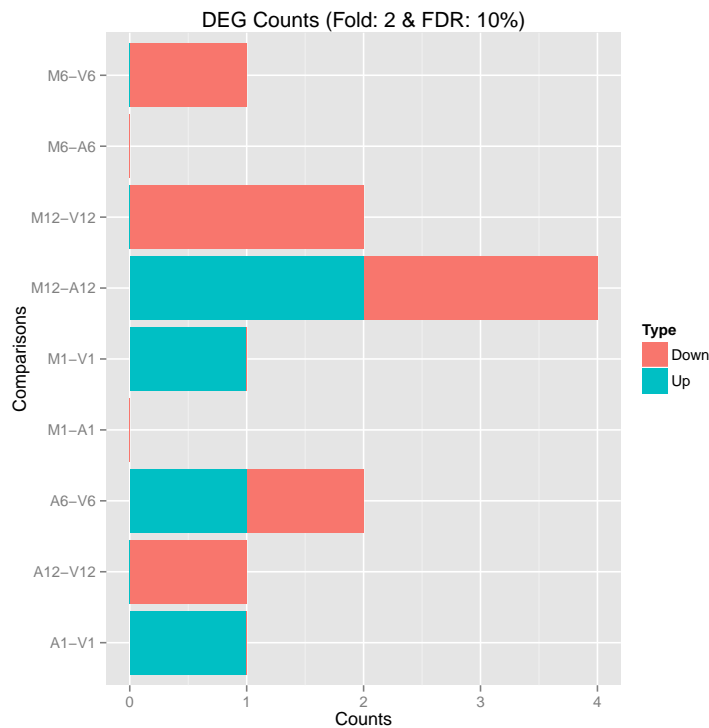


Figure 3: Up and down regulated DEGs with FDR of 1%.

The function `overLapper` can compute Venn intersects for large numbers of sample sets (up to 20 or more) and `vennPlot` can plot 2-5 way Venn diagrams. A useful feature is the possibility 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()
```

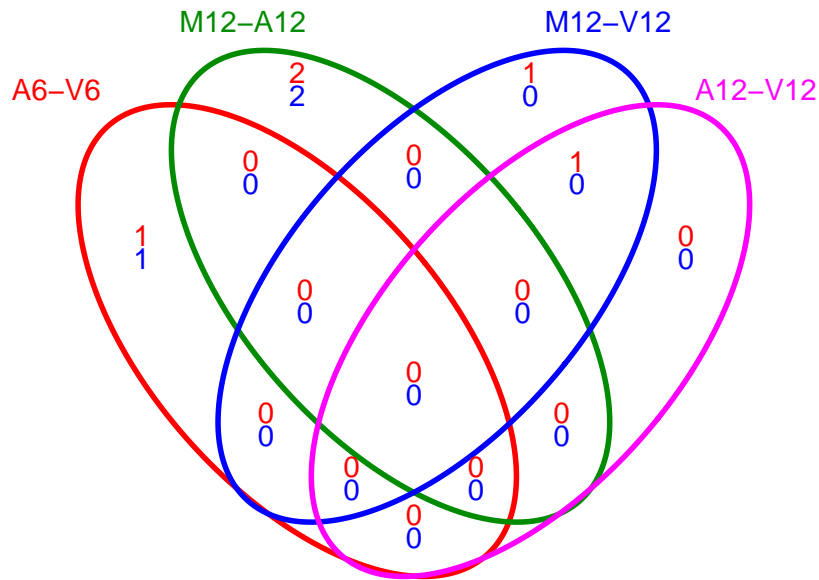


Figure 4: Venn Diagram for 4 Up and Down DEG Sets.

## 6.1 GO term enrichment analysis of DEGs

### 6.1.1 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
m <- useMart("ENSEMBL_MART_PLANT"); listDatasets(m)
m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
listAttributes(m) # Choose data types you want to download
go <- getBM(attributes=c("go_accession", "tair_locus", "go_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]=="cellul
go[1:4,]
dir.create("./data/GO")
write.table(go, "data/GO/GOannotationsBiomart_mod.txt", quote=FALSE, row.names=FALSE, col.names=FALSE, sep=
catdb <- makeCATdb(myfile="data/GO/GOannotationsBiomart_mod.txt", lib=NULL, org="", colno=c(1,2,3), idconv=
save(catdb, file="data/GO/catdb.RData")
```

### 6.1.2 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 `GOCluster_Report` 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.

```
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, cutoff=
library("biomaRt"); m <- useMart("ENSEMBL_MART_PLANT", dataset="athaliana_eg_gene")
goslimvec <- as.character(getBM(attributes=c("goslim_goa_accession"), mart=m)[,1])
BatchResultslim <- GOCluster_Report(catdb=catdb, setlist=DEGlist, method="slim", id_type="gene", myslimv=g
```

### 6.1.3 Plot batch GO term results

The data.frame generated by `GOCluster_Report` 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")
```

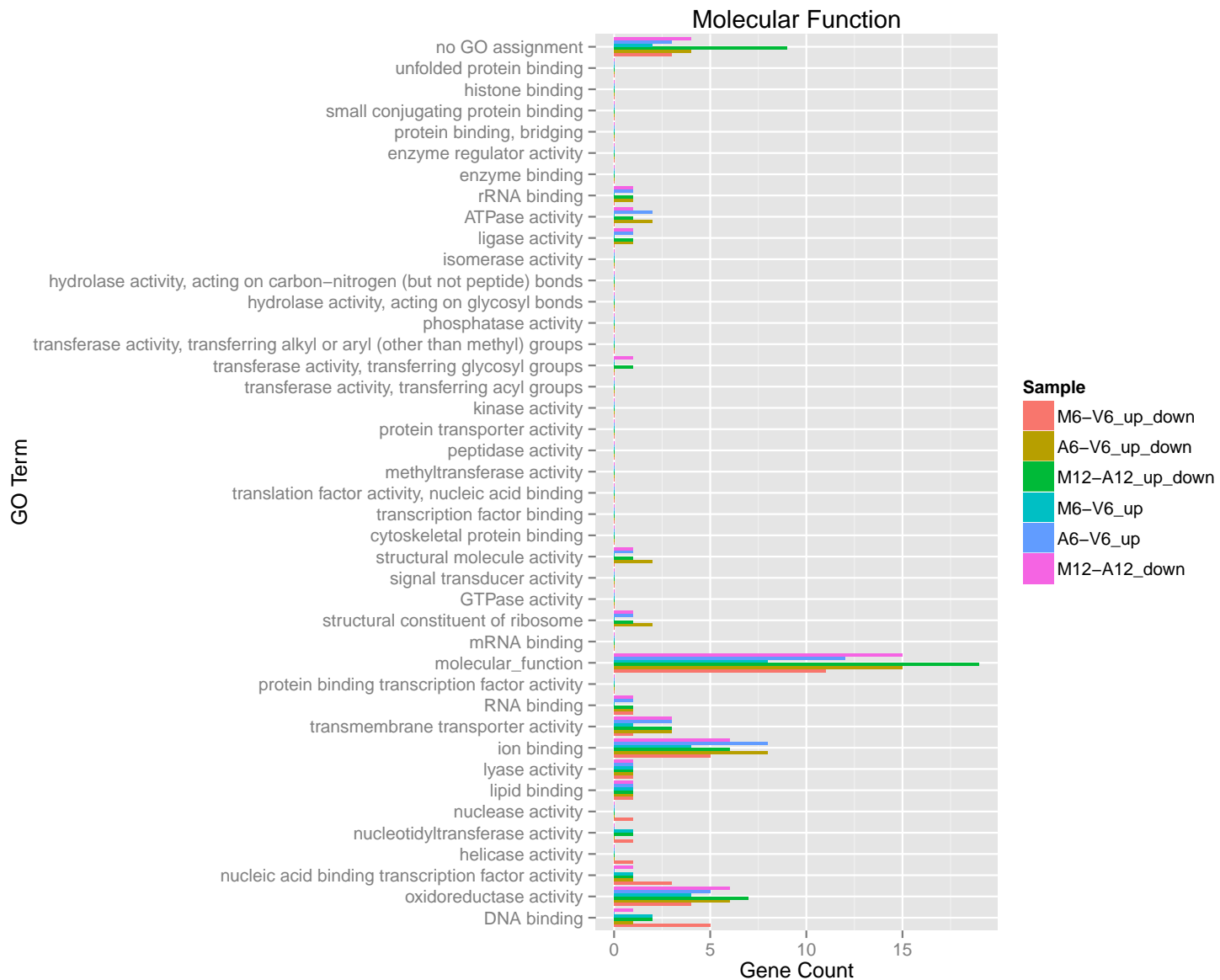


Figure 5: GO Slim Barplot for MF Ontology.

## 7 Clustering and heat maps

The following example performs hierarchical clustering on the `rlog` transformed expression matrix subsetting 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()
```

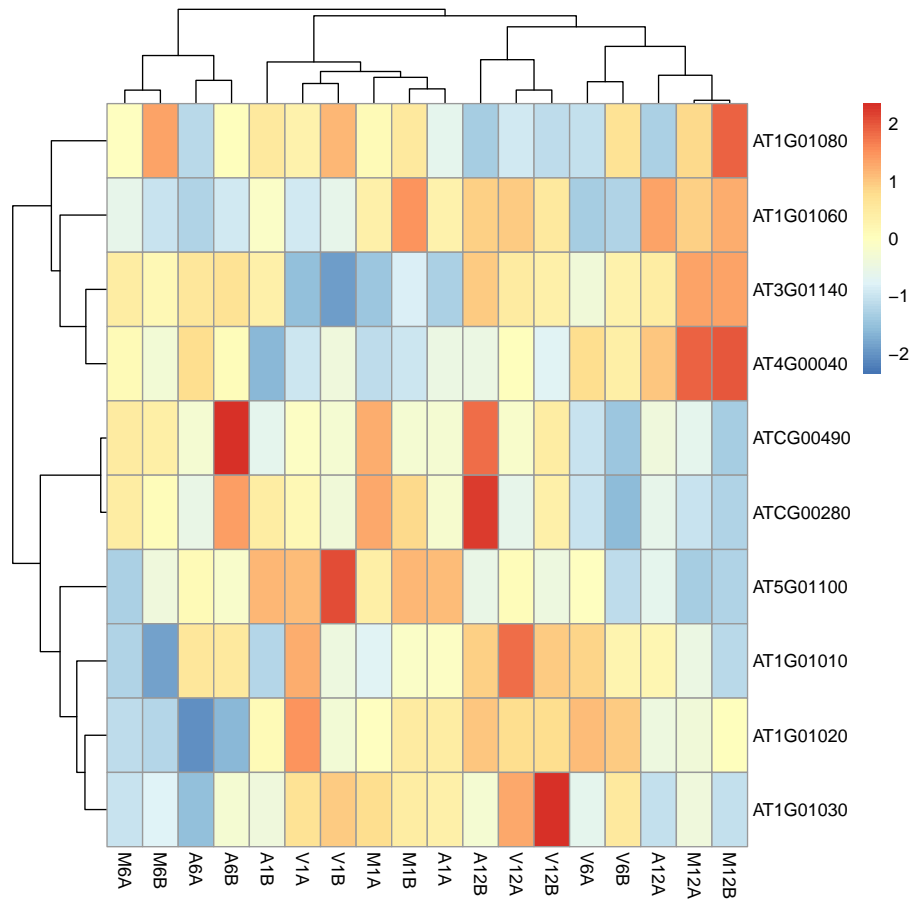


Figure 6: Heat map with hierarchical clustering dendrograms of DEGs.

## 8 Version Information

```
toLatex(sessionInfo())
```

- R version 3.3.1 (2016-06-21), x86\_64-apple-darwin13.4.0
- Locale: C/en\_US.UTF-8/en\_US.UTF-8/C/en\_US.UTF-8/en\_US.UTF-8
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.32.0, BiocGenerics 0.18.0, BiocParallel 1.6.4, BiocStyle 2.0.3, Biostrings 2.40.2, DESeq2 1.12.3, GenomInfoDb 1.8.3, GenomicAlignments 1.8.4, GenomicRanges 1.24.2, IRanges 2.6.1, Rsamtools 1.24.0, S4Vectors 0.10.2, ShortRead 1.30.0, SummarizedExperiment 1.2.3, XVector 0.12.1, ape 3.5, ggplot2 2.1.0, knitr 1.13, systemPipeR 1.6.4
- Loaded via a namespace (and not attached): AnnotationDbi 1.34.4, AnnotationForge 1.14.2, BBmisc 1.10, BatchJobs 1.6, Category 2.38.0, DBI 0.4-1, Formula 1.2-1, GO.db 3.3.0, GOstats 2.38.1, GSEABase 1.34.0, GenomicFeatures 1.24.5, Hmisc 3.17-4, Matrix 1.2-6, RBGL 1.48.1, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.6, XML 3.98-1.4, acepack 1.3-3.3, annotate 1.50.0, backports 1.0.3, base64enc 0.1-3, biomaRt 2.28.0, bitops 1.0-6, brew 1.0-6, checkmate 1.8.1, chron 2.3-47, cluster 2.0.4, codetools 0.2-14, colorspace 1.2-6, data.table 1.9.6, digest 0.6.10, edgeR 3.14.0, evaluate 0.9, fail 1.3, foreign 0.8-66, formatR 1.4, genefilter 1.54.2, geneplotter 1.50.0, graph 1.50.0, grid 3.3.1, gridExtra 2.2.1, gtable 0.2.0, highr 0.6, htmltools 0.3.5, hwriter 1.3.2, labeling 0.3, lattice 0.20-33, latticeExtra 0.6-28, limma 3.28.17, locfit 1.5-9.1,

magrittr 1.5, munsell 0.4.3, nlme 3.1-128, nnet 7.3-12, pheatmap 1.0.8, plyr 1.8.4, rjson 0.2.15, rmarkdown 1.0, rpart 4.1-10, rtracklayer 1.32.2, scales 0.4.0, sendmailR 1.2-1, splines 3.3.1, stringi 1.1.1, stringr 1.0.0, survival 2.39-5, tools 3.3.1, xtable 1.8-2, yaml 2.1.13, zlibbioc 1.18.0

## 9 Funding

---

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