Introduction to R

Anindya Bhattacharya, PhD | abhatta3@ucla.edu
CBI fellow | Xia Yang lab

Workshop 3: Introduction to R

- **Day 3**
  - Automating or scripting R
    - Batch mode
  - Packages that extend R
    - Installation
    - Documentation
    - Useful packages
  - Bioconductor
    - Extending R for the life sciences
    - Useful bioconductor packages
    - Some basic bioC concepts
    - Bioconductor demo

- Future Workshop Enrollment

Did anyone get bioconductor to install?

A few more useful commands

- merge(df1, df2, by.x=1, by.y=1)
- subset()
- identify()
- paste()

Batch processing in R

- Suppose you have an R script that reads in data, analyzes it and outputs the result
- You can use that script to process multiple files
  - e.g. you have 100 files to process in an identical fashion
  - e.g. you want to generate 50 plots
- You might have a long running R task
  - you can script the processing and leave it running overnight and return the next day to examine the results

R command line options

- The help on R (eg man R) shows R has several command line options useful for batch

```
--no-save
--no-restore
--no-init-file
--vanilla (combines all the above plus more)
--slave (makes R talk less)
```

Batch Mode

- Suppose you have a file batchscript1.R that contains

```
df = data.frame(id = 1:10, data=rnorm(10))
cat("begin processing\n")
output = dim(df)
output
cat("end processing\n")
```

- Invoke the script as:

```
R CMD BATCH --vanilla --slave batchscript1.R output.txt
```

- This will run the commands in batchscript1.R
- It will write the output to output.txt

Batch mode

- Another way to doing the same thing

```
R --vanilla --slave < input.R > output.txt
```

- input.R is the input file and all output will be written (redirected) to output.txt

Batch scripts can take arguments

- example script batchscript2.R:

```
args = commandArgs(TRUE)
a = paste(args, collapse="")
cat(a, "\n")
cat("reading file: ", args[1], "\n")
# df = read.table(args[1])
cat("begin processing\n")
output = dim(df)
output
cat("end procesing\n")
```

- Invoke

```
R CMD BATCH --vanilla --slave '--args file1' batchscript2.R batchscript2.Rout
```

Batch mode example

- R script example: Following R script take two arguments as input and then check their values
- Type following R instructions in a new file (you can use any text editor) then save file as Example.r

```
args <- commandArgs(TRUE)
pval_thr=as.numeric(args[1])
filtertype=args[2]

if( pval_thr>0.5 ) {
   cat("Correlation Thr = ",pval_thr,"\n")
}  else {
  cat("Correlation Thr too low = ",pval_thr,"Use value higher than 0.5\n")
}
if(filtertype=="M") {
 cat("You can also select L and H\n")
} else {
cat("You have seleced filter type = ",filtertype,". You can also select M\n")
}
```

- Place file in your R script directory.
- cd to R script directory.
- Run following instruction from Linux

```
R CMD BATCH --vanilla --slave '--args 0.6 M' Example.r  output.txt
```

- Look for output in output.txt file

Why Call C or Fortran from R?

- Loops are slow in R
- **Solution: C functions and Fortran subroutines callable from R**

```
void foo(int *nin, double *x) {
int n = nin[0];
int i;
for (i=0; i<n; i++)
   x[i] = x[i] * x[i];
}
```

Compiling and Dynamic Loading

- Put the C example code in a file foo.c and compile it to a shared library. The command (in UNIX)

$R CMD SHLIB foo.c

- Now the code can be dynamically loaded into R by doing (in R)

> dyn.load("foo.so")

The Call to C

- The actual call to C from R is made by the R function .C like this

```
>.C("foo", n=as.integer(5), x=as.double(rnorm(5)))
```

R package

- To start a package for our R code all we have to do is run function package.skeleton() and pass it the name of the package we want to create plus a list of all source code les.

```
> package.skeleton(name="linmod", code_files="linmod.R")
Creating directories ...
Creating DESCRIPTION ...
Creating Read-and-delete-me ...
Copying code files ...
Making help files ...
Done.
Further steps are described in './linmod/Read-and-delete-me'.
```

File Read-and-delete-me

- Edit the help file skeletons in 'man', possibly
- Combining help files for multiple functions.
- Put any C/C++/Fortran code in 'src'.
- If you have compiled code, add a .First.lib() function in 'R' to load the shared library.
- * Run R CMD build to build the package tarball.
- * Run R CMD check to check the package tarball.

The package DESCRIPTION file

- Package: MyPkg
- Type: Package
- Title: What the package does (short line)
- Version: 1.0
- Date: 2014-06-04
- Author: Who wrote it
- Maintainer: Who to complain to <yourfault@somewhere.net>
- Description: More about what it does (maybe more than one line)
- License: What license is it under?

Package from R studio

- 'File' menu-> 'New Project' -> 'New Directory' -> 'R Package'
- Fill information
  - Package name
  - Location.
- 'Create Project' button.

Installing from Bioconductor

- set of packages tailored to life sciences
- requires its own installation method but relies on many tools from CRAN
- Navigating the website

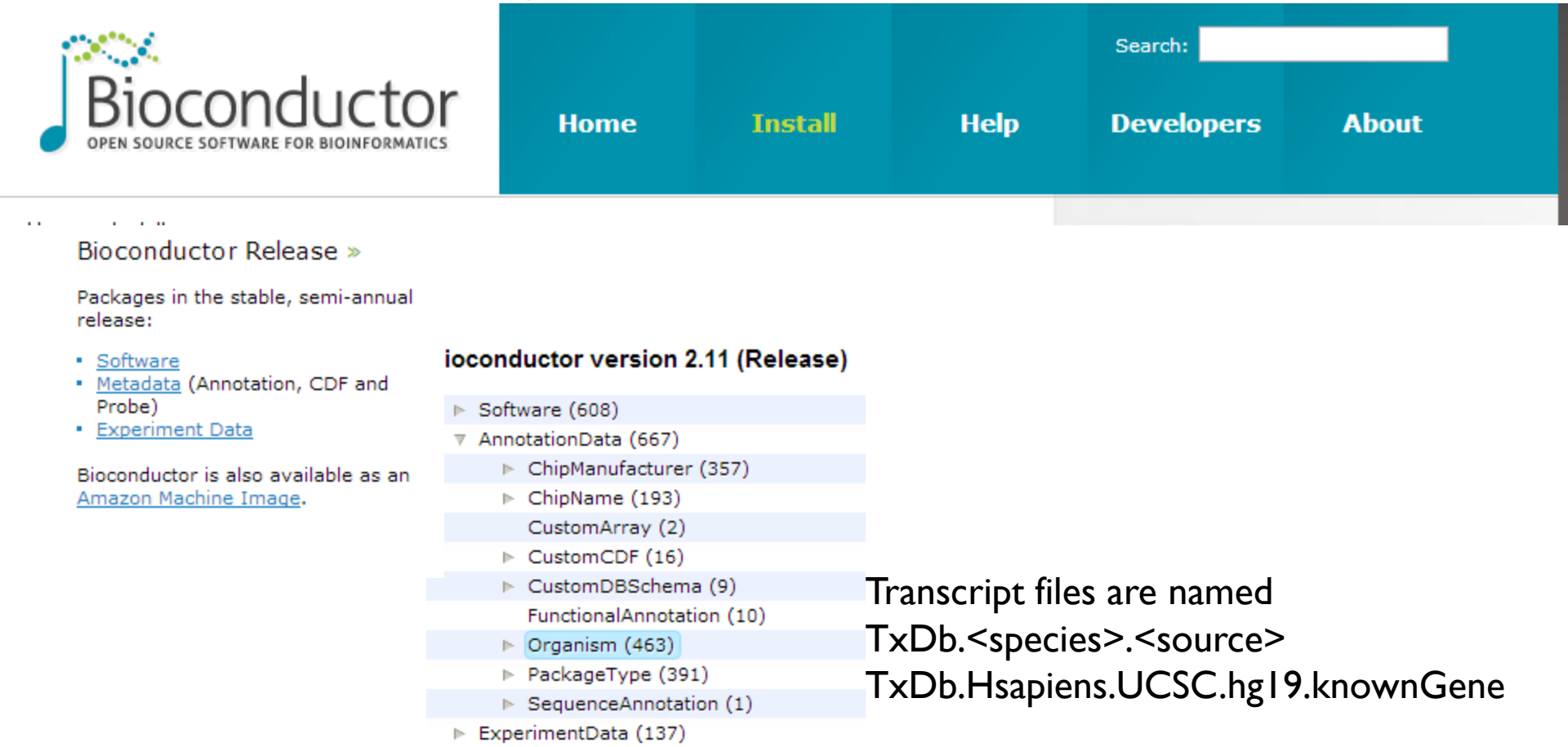
```
source("http://bioconductor.org/biocLite.R")
biocLite()
biocLite("DESeq")
```

Bioconductor

- Uses an OOP methodology (S4 classes)
  - not trivial but not necessary to understand details to use
- functionality is tied up in classes, which have methods
- methods allow
  - setting the data value of a class instance
  - access the data value of a class instance
  - manipulation of data (eg normalization, etc)
- Each BioC package has a Vignette and documentation

Where to find annotation

- Bioconductor.org



Transcript files are named
TxDb.<species>.<source>
TxDb.Hsapiens.UCSC.hg19.knownGene

Affy package

- Installation
  - source("http://bioconductor.org/biocLite.R")
  - biocLite("affy")
- The Affy package provides basic methods for analyzing affymetrix oligonucleotide arrays.
  - **Obtaining scaled expression values with 5 different methods** (MAS5, RMA, GCRMA, Plier & dChip).
    - library(affy)
    - Reads all *.CEL (*.cel) files in your current working directory and stores them into the AffyBatch object 'mydata'.
      >mydata <- ReadAffy()
    - Opens file browser to select specific CEL files.
      > mydata <- ReadAffy(widget=TRUE)
    - Creates normalized and background corrected expression values using the RMA method.
      > eset <- rma(mydata)

Writes expression values to text file

- > write.exprs(eset, file="mydata.txt")

Annotation data for Affy IDs

- Opens library with annotation data.

```
>library(ath1121501.db)
```

- Shows availability and syntax for annotation data.

```
>library(help=ath1121501.db)
```

- Provides a summary about the available annotation data sets of an annotation library.

```
>ath1121501()
```

- Retrieves all Affy IDs for a chip in vector format.

```
>library(ath1121501cdf);  ls(ath1121501cdf)
```

Chromosome maps

- Displays all genes on the chromosomes of an organisms.

Genes encoded by the - strands are represented by lines below the chromosomes.

```
> library(annotate); library(geneplotter);
  library(hgu95av2); newChrom <-
  buildChromLocation("hgu95av2"); newChrom;
  cPlot(newChrom)
```

- This highlights in the above plot a set of genes of interest in red color (e.g. expressed genes of an experiment).

```
> data(sample.ExpressionSet); myeset <-
  sample.ExpressionSet;
  cColor(featureNames(sample.ExpressionSet), "red",
  newChrom)
```

Basic usage of GO information

```
> library(GOstats); library(GO.db); library(ath1121501.db);
  library(annotate)
```

GO similarity

```
> library(GOSemSim)
> help(GOSemSim)
```

```
>goSim("GO:0004022", "GO:0005515", ont = "MF",
   measure = "Wang")
[1] 0.158
> go1 = c("GO:0004022", "GO:0004024", "GO:0004174")
> go2 = c("GO:0009055", "GO:0005515")
> mgoSim(go1, go2, ont = "MF", measure = "Wang",
   combine = NULL)
```

DESeq

- Run through the demo
- Memory issues, compute issues
  - size of data
  - memory of machine, disk storage
  - may require using a cluster (hoffman2), cloud (AWS)

CummeRbund

- bioconductor package for examining Cufflinks output
- only works with versions of Cufflinks version 2 or newer
- run through the demo

NATURE PROTOCOLS | PROTOCOL

Differential gene and transcript expression analysis of RNA-seq experiments with TopHat and Cufflinks

Cole Trapnell, Adam Roberts, Loyal Goff, Geo Pertea, Daehwan Kim, David R Kelley, Harold Pimentel, Steven L Salzberg, John L Rinn & Lior Pachter

Affiliations | Contributions | Corresponding author

Nature Protocols **7**, 562–578 (2012) | doi:10.1038/nprot.2012.016
Published online 01 March 2012

Class Survey

- https://www.surveymonkey.com/s/2YD7ZXP

Anindya Bhattacharya, PhD

abhatta3@ucla.edu

Office hours: 2000 TLSB

Xia Yang lab

Department of Integrative Biology and Physiology

Areas of interest: miRNA, genetic variation, gene expression data analyses, machine learning,

Collaboratory Website

http://collaboratory.lifesci.ucla.edu/