Key R commands:

Download the latest version of R for your machine at: http://cran.at.r-project.org/

A useful resource for using R for Bioinformatics can be found at:
http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual

Basics & help:

Double click icon to Open R

help.start() #launches HTML help system
help.search("topic") #searches online help files for topic
?function #loads help on any given function of R or loaded package
help(function) #same
example(function) #gives an example of a functions use
q() #quit R console

To install packages:

For Bioconductor, cut & paste these lines: (note that # is a comment indicator, so everything on a line after # is ignored by R).

source("http://bioconductor.org/biocLite.R") # Sources the biocLite.R installation script.
biocLite() # Installs the biocLite.R default set of BioConductor packages.
if desired…
biocLite("pkg_name_1")
biocLite(c("pkg1", "pkg2")) # Command to install additional packages from BioC.

Use this occasionally to update Bioconductor packages…
update.packages(repos=biocinstallRepos(), ask=FALSE) # Bioconductor packages, especially those in the development branch, are updated fairly regularly. To update all of your installed packages, run this command after sourcing "biocLite.R".

To install other packages, use menus: Packages>Install Packages, choose a mirror site & select the desired package. Alternatively use:
install.packages() #to install or…
update.packages() # to update.

Managing Packages & scripts:

library() # tells which packages are installed
library(package_name) #loads a package for use
search() #tells which packages are currently loaded
library(help=package_name) #lists all functions in a package

source("my_script") # Command to execute an R script, here 'my_script'. For example, generate a text file 'my_script' with the command 'print(1:100)', then execute it with the source function.

Basic R functions:

ls() # lists current objects, same as objects()
rm(object1,object2,…) # removes objects
rm(list=ls()) # removes all objects w/o warnings
gc() # garbage collection to clean up memory
summary(object) # Generic summary info for all kinds of objects.
attributes(object) # Returns an object's attribute list.
dim(object) #returns the dimensions of vectors, arrays or dataframes

name_of_any_object #lists contents of that object
new_object<-function(object) #performs function on object & puts results in new_object
obj1<-function(obj1, numeric_parameter=#, text_param="text") #other parameters in a function are assigned by "parameter_name=" statements, separated by commas). Text must be quoted. Warning – smart " " quotes from MSWord etc, will not be read properly.
c(#,"text","text2",#2) #catenates different inputs together into a list. Sometimes shows up in function calls referring to multiple objects for a single parameter.
You can write several commands per line, separated by semicolons - ;
If a line is unfinished (e.g. command won't run due to final bracket) can type return & continue on 2nd line (after + symbol)

dir() #lists content of current working directory
getwd() #prints current working directory
setwd("c:/folder/folder2") #set working directory. Beware that "\" is considered a special character in R. Pathnames that use "\" separators need to be changed to "\\" or "/"

Inporting data from keyboard or files:

x <- edit(data.frame()) # Starts empty GUI spreadsheet editor for manual data entry.
x <- edit(x) # Opens existing data frame (table) 'x' in GUI spreadsheet editor.
read.delim("clipboard", header=T) # Command to copy&paste tables from Excel or other programs into R. If the 'header' argument is set to FALSE, then the first line of the data set will not be used as column titles.
my_frame <- read.table(file="my_table", header=TRUE, sep="\t") # Reads a table & assigns it to a dataframe, specifying tab separated & with first row as headers. To import in "character mode" include colClasses="character".
my_frame<-read.delim("file", na.strings ="", fill=TRUE,header=T,sep="\t") #alternative read method better if empty fields exist, or if strings are very long
file.show("file_name") #prints contents of file to screen, allowing scrolling
scan("my_file") #makes vector from single-column file data
vector<-**as.vector**(single_column_table) # may be useful if reading in a single column of data, to convert it to a vector
as.character(vector) # may be useful to allow effective matches to character-type data

Exporting data from objects to files:

write.table(object_name, "clipboard", sep="\t", col.names=NA, quote=F) # Command to copy&paste from R into Excel or other programs. It writes the data of an R data frame object into the clipbroard from where it can be pasted into other applications (limited buffer size).
write.table(my_frame, file="my_file", sep="\t", col.names = NA) # Writes data frame to a tab-delimited text file. The argument 'col.names = NA' makes sure that the titles align with columns when row/index names are exported (default). row.names = FALSE will suppress leftmost column of row names (default = T). quote = FALSE will remove "" marks from around all text items (default=T).
save(x, file="my_file.txt"); load(file="file.txt") # Commands to save R object to an external file and to read it into R again from this file.

Manipulating data frames, matrices & vectors:

Subsetting rules:
array[row, column, addit_dimension] #how to refer to each cell in an array
matrix[1:4,] #refers to all columns & 1st 4 rows
matrix[-c(1:4),1] #refers to all but 1st 4 rows in column 1 (by using negative sign)
data.frame()
frame<-**data.frame**("col_title1"=vector, "col_title2"=vector2) #creates 2 column data frame
frame<-data.frame(frame, "col_title3"=matrix[,1]) #appends column 1 from matrix to frame
frame<-data.frame(frame, "col_title4"=matrix2[vector,1]) #appends only those elements of column 1 in matrix 2 that are in vector to frame
colnames(frame) #prints all column names
colnames(frame)[3]<-"newname" #changes the name of the 3rd column only
rownames(frame)<-vector #assigns rownames from vector to rows of frame
order()
ordered_frame<-frame[**order**(frame$col_title3),] sorts elements in frame by values in column 3. Can also add additional sort keys by order(1st_key,2nd_key,3rd_key)…
t(matrix_or_frame) # transposes x<->y (rows become columns & vice versa)
unique(vector_name) #returns only the unique elements in a vector (or vector from indexed frame column) or the unique complete rows in a frame or matrix
intersect(vector,table[,2]) #returns only values present in both vectors: here "vector" & the vector in table col 2
%in%...
frame2<-frame1[frame1$COL_NAME1 **%in%** "Item_to_search_for",] #outputs frame with only those rows that have "Item_to_search_for" as the complete content of COL_NAME1.
frame2<-frame1[frame1$COL_NAME1 **%in%** vector,] #keeps only rows where COL_NAME1 matches any entry in "vector"
merge(frame1,frame2,by.x
="COL_NAME_IN_FRAME1",by.y="COL_NAME_IN_FRAME_2",all=FALSE) #finds all rows where entries match in column names specified by.y & by.x & outputs new frame including only columns from both frames where entries matched. If all=TRUE, outputs all rows of both frames, including unmatched & using NAs where match didn't occur.

cbind(vector1,vector2) #binds columns of the same length together as a matrix or frame. Can use cbind(frame[3],frame[1]) etc. to select & reorder columns in an existing frame
rbind(vector1,vector2) #as cbind, but attaching vectors together as rows in matrix or frame.

Manipulating text and data elements:

paste("text",as.character(number),"text2", sep="") #concatenates entries together into a single text entry w/ no spaces. Use sep="seperator" to separate by spaces or chosen string.
vector_out<-**paste**(vector,vector,column_in_frame,sep=":") #performs paste iteratively on each element in the vectors & puts results in vector_out. Warning, if vectors are of different length will cycle back through the shorter ones!
sub("pattern","replacement",object) #replaces all instances of pattern contained (not across) entries with replacement. Can use special wildcard characters in search pattern. For instance "chr1:.*" finds any field that starts "chr1:" followed by any number of additional characters. For more info, look up regexpr rules.
grep("text_pattern", vector, value=T) #finds & outputs all instances of text_pattern in vector (which could be a column in a frame by using frame$COL_NAME). If value=F (default) just reports number of instances. See ?grep for using regexp characters to search for things other than exact strings.

Bioconductor Examples:

Analysis of Affy .CEL files with GCRMA followed by comparisons with Limma: Note: these packages come with their own defined functions, which you need to read the manual to understand. These package-specific functions are in bold, below.

```
> library(gcrma); library(limma) #you can put several commands on a line, separated by a semicolon
>  targets<-readTargets("targets_w_Katrin4h.txt")
> ab<-ReadAffy(filenames=targets$filename) #all files should be in home directory, as set above
> eset<-gcrma(ab)
> treatments<-factor(c(1,1,1,2,2,2,3,3,3,3,3,3,4,4,4,4,4),labels=c("E176_E2","E176_veh","WT_E2","WT_Veh"))
> design<-model.matrix(~0+treatments)
> fit<-lmFit(eset,design)
> colnames(design)<-c("E176_E2","E176_veh","WT_E2","WT_Veh")
> contrast.matrix<-makeContrasts(WT_E2-WT_Veh,E176_E2-E176_veh,E176_veh-WT_Veh,E176_E2-
    WT_E2,levels=design) #lists comparisons to make: WT_E2-WT_Veh looks at expression ratios of E2/Veh
> fit2<-contrasts.fit(fit,contrast.matrix) #linear fit
> fit3<-eBayes(fit2) #Bayes analysis to handle standard deviations better
> toptable(fit3, adjust.method="fdr", number=150, coef=1)
#coef=1 means to list the first comparison (WT_E2-WT_Veh)
#adjust.method="fdr" sets method for calculating adjusted P values. Works as well as the default = "BH". Others are
    more stringen.
#number=150, sets number of lines to print out (default ~20)
```

Contents of "targets_w_Katrin4h.txt:

Scan.Name	Sample	filename	Sample.1
MM2007060739	E176 mAo 1 +E2	MM2007060739.CEL	E176E2
MM2007060743	E176 mAo 2 +E2	MM2007060743.CEL	E176E2
MM2006121258	E176 mouse Ao +E2	MM2006121258.CEL	E176E2
MM2007060741	E176 mAo 1 +Veh	MM2007060741.CEL	E176Veh

MM2007060745	E176 mAo 2 +Veh	MM2007060745.CEL	E176Veh
MM2007060747	E176 mAo 3 +Veh	MM2007060747.CEL	E176Veh
MM2007060740	WT mAo 1 +E2	MM2007060740.CEL	WTE2
MM2007060744	WT mAo 2 +E2	MM2007060744.CEL	WTE2
MM2006121257	WT mouse Ao +E2	MM2006121257.CEL	WTE2
MB2006063034	10-Estrogen 4 hr (Katrin)	MB2006063034.CEL	WTE2
MB2006063035	11-Estrogen 4 hr (Katrin)	MB2006063035.CEL	WTE2
MB2006063036	12-Estrogen 4 hr (Katrin)	MB2006063036.CEL	WTE2
MM2007060742	WT mAo 1 +Veh	MM2007060742.CEL	WTVeh
MM2007060746	WT mAo 2 +Veh	MM2007060746.CEL	WTVeh
MM2007060748	WT mAo 3 +Veh	MM2007060748.CEL	WTVeh
MB2006063037	13-vehicle 4 hr (Katrin)	MB2006063037.CEL	WTVeh
MB2006063038	14-vehicle 4 hr (Katrin)	MB2006063038.CEL	WTVeh

Use of Affy program instead of GCRMA to read .CEL files, normalize & get QC info:

Package-specific functions are in bold, below.

> library(gcrma); library(limma) #you can put several commands on a line, s