同濟大學
TONGJI UNIVERSITY

生物信息学系
DEPARTMENT OF BIOINFORMATICS

Section 10

R: Basics

张勇

yzhang@tongji.edu.cn

同济大学

2021年5月17日

Why R Cookbook?

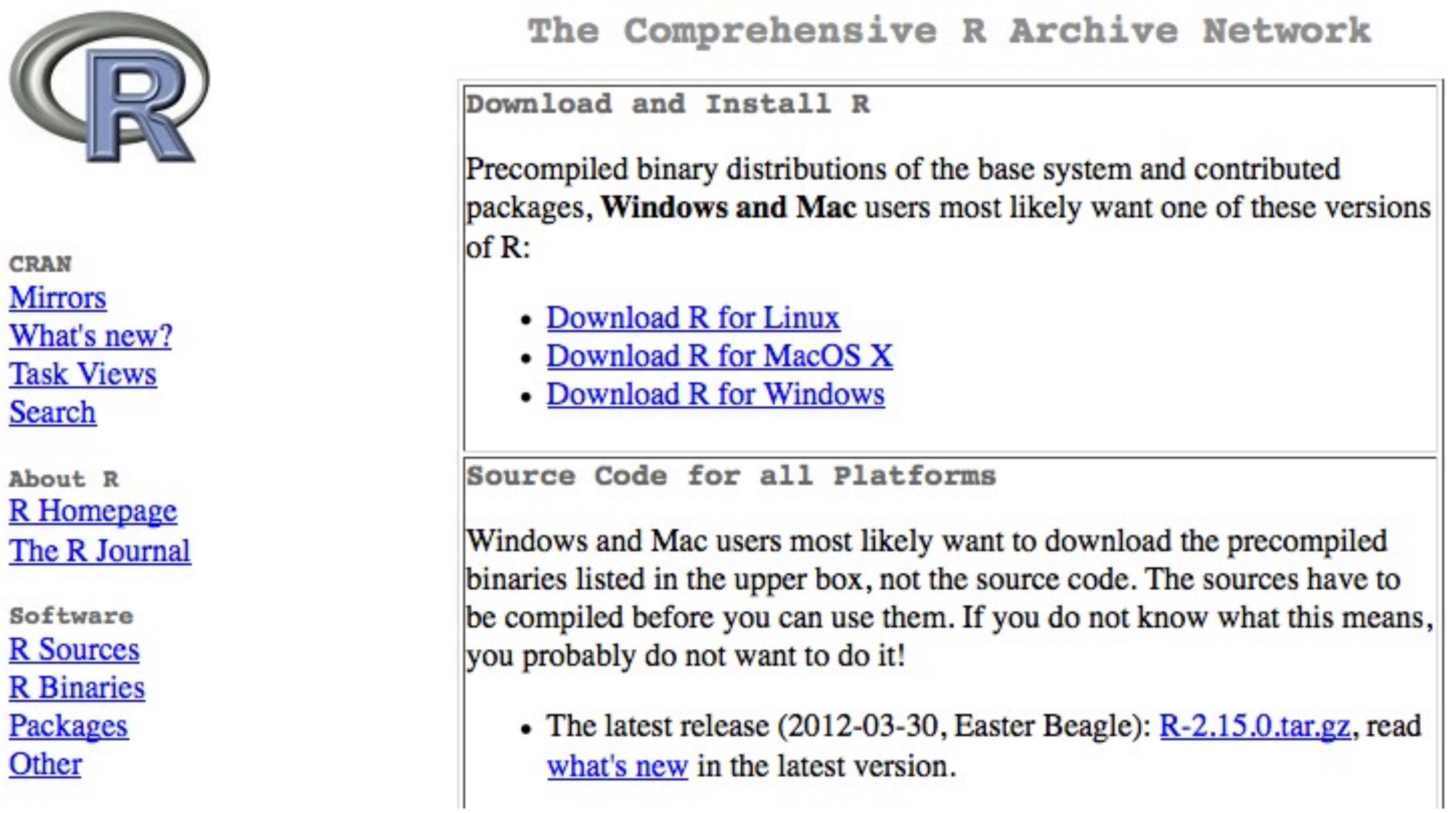
- R is a powerful tool for statistics, graphics, and statistical programming.
  - Tens of thousands of users daily.
  - Free, open source system.
  - More than 14,000 available add-ons.
  - A serious rival to all commercial statistical packages.
- R can be frustrating.
  - It’s not obvious how to accomplish many tasks, even simple ones.
  - The simple tasks are easy once you know how, yet figuring out that “how” can be maddening.

Downloading and Installing R

- **Problem**
  - You want to install R on your computer.

- **Solution**
  - Windows and OS X users can download R from CRAN (Comprehensive R Archive Network). Linux and Unix users can install R packages using their package management tool.
  - http://cran.r-project.org/

Downloading and Installing R



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for MacOS X
- Download R for Windows

Source Code for all Platforms

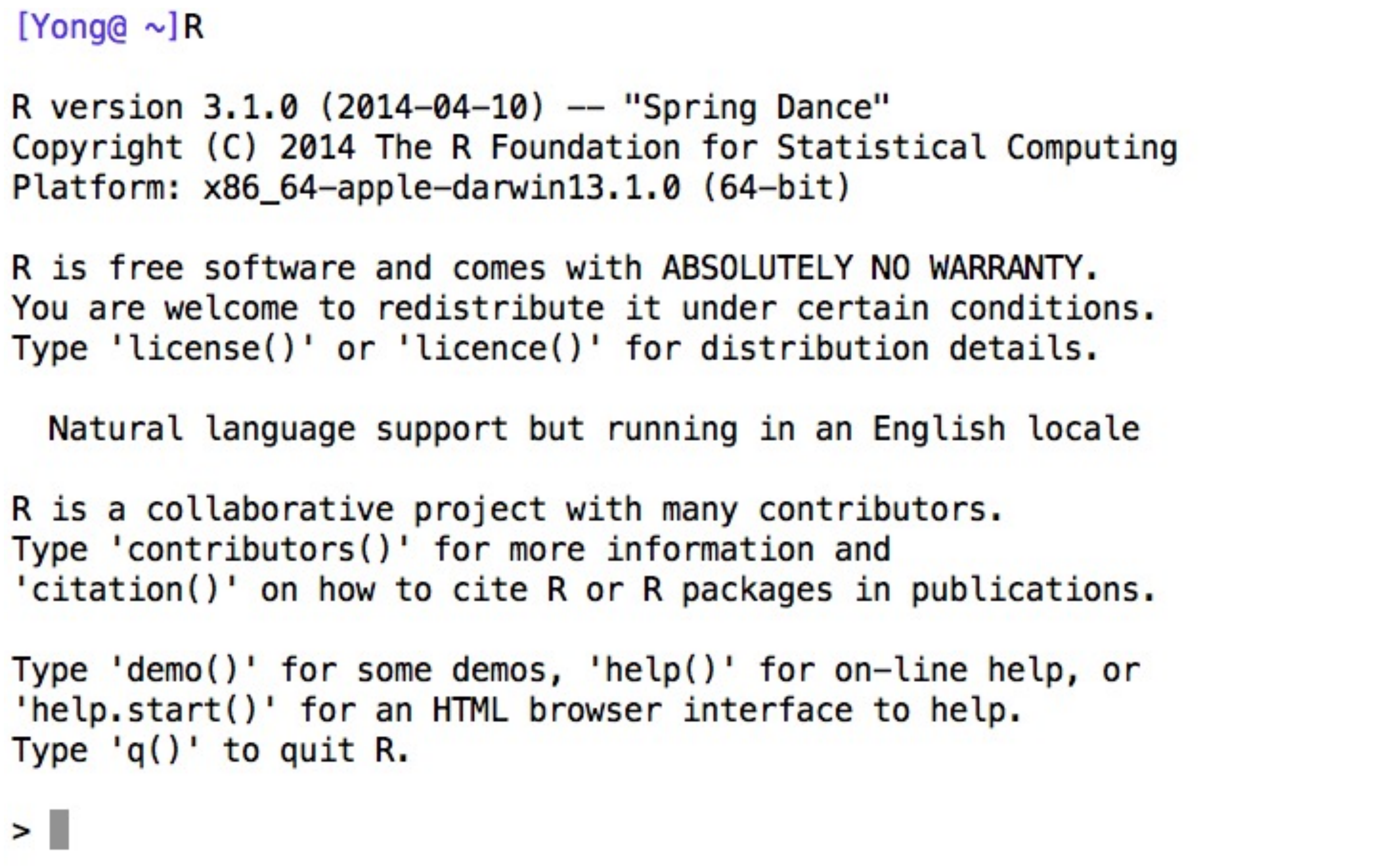
Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2012-03-30, Easter Beagle): R-2.15.0.tar.gz, read what's new in the latest version.

Starting R

- **Problem**
  - You want to run R on your computer.
- **Solution**
  - Windows: Click on Start -> All Programs -> R;
  - OS X: Click on the icon in the Applications directory; type **R** on a Unix command
  - Linux or Unix: Start the R program from the shell prompt using the **R** command

Starting R



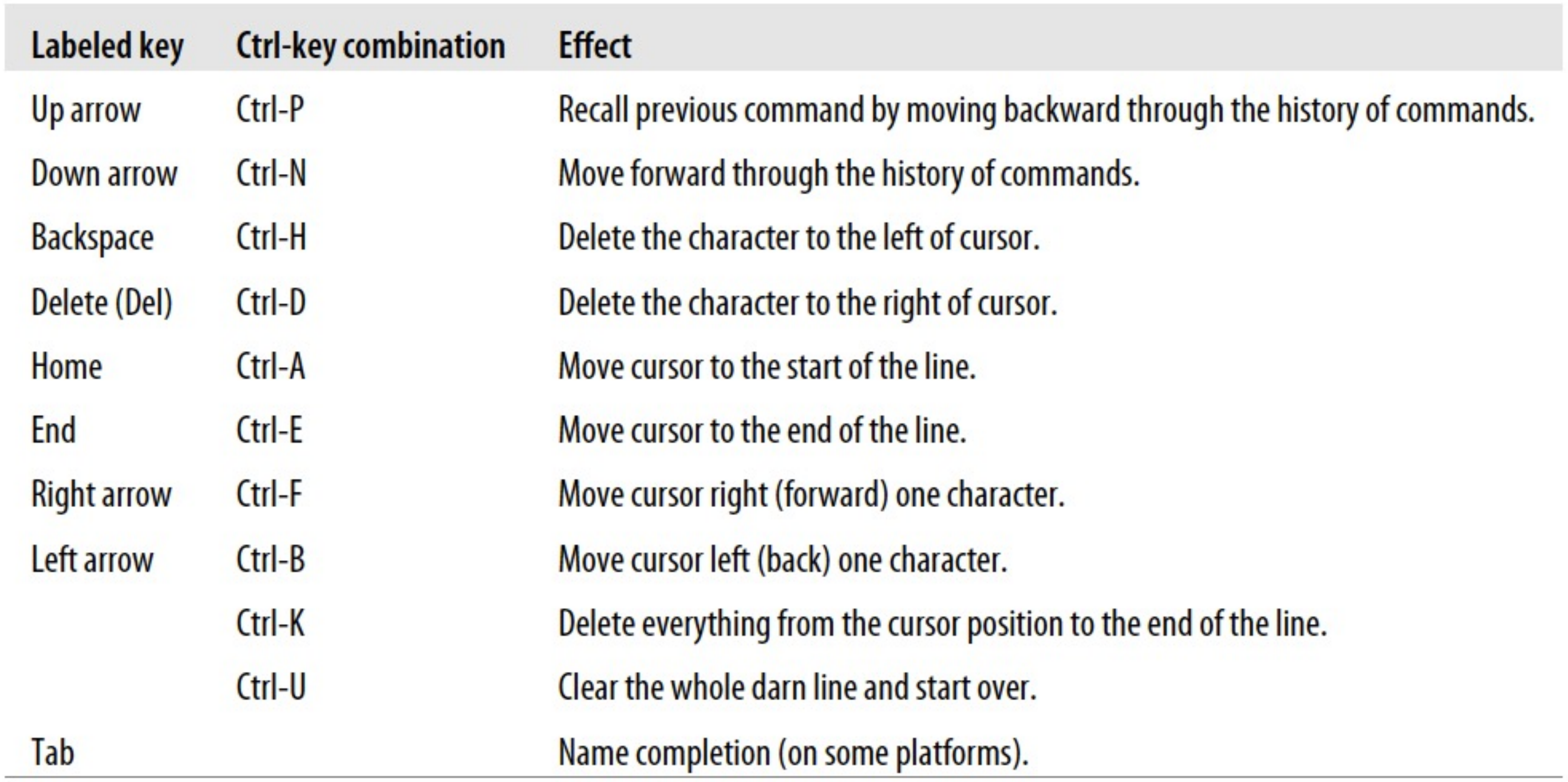
```
[Yong@ ~]R

R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.1.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Entering Commands

- **Problem**
  - You’ve started R. Now what?
- **Solution**
  - Simply enter expressions at the command prompt. R will evaluate them and print display the result.

```
> 1 + 1
[1] 2
> max(1, 2, 3)
[1] 3
```

Entering Commands



Labeled key	Ctrl-key combination	Effect
Up arrow	Ctrl-P	Recall previous command by moving backward through the history of commands.
Down arrow	Ctrl-N	Move forward through the history of commands.
Backspace	Ctrl-H	Delete the character to the left of cursor.
Delete (Del)	Ctrl-D	Delete the character to the right of cursor.
Home	Ctrl-A	Move cursor to the start of the line.
End	Ctrl-E	Move cursor to the end of the line.
Right arrow	Ctrl-F	Move cursor right (forward) one character.
Left arrow	Ctrl-B	Move cursor left (back) one character.
	Ctrl-K	Delete everything from the cursor position to the end of the line.
	Ctrl-U	Clear the whole darn line and start over.
Tab		Name completion (on some platforms).

Keystrokes for command-line editing

Exiting from R

- **Problem**
  - You want to exit from R.

- **Solution**
  - Linux or Unix: press Ctrl-D.
  - Use the **q** function to terminate R on all platforms.

```
> q()
Save workspace image? [y/n/c]:
```

Note: If you save your workspace, R writes it to a file called *.RData* in the current working directory.

Interrupting R

- **Problem**
  - You want to interrupt a long-running computation and return to the command prompt without exiting R.

- **Solution**
  - Linux or Unix: Press Ctrl-C. This will interrupt R without terminating it.

  Note: Interrupting R can leave your variables in an indeterminate state, depending upon how far the computation had progressed.

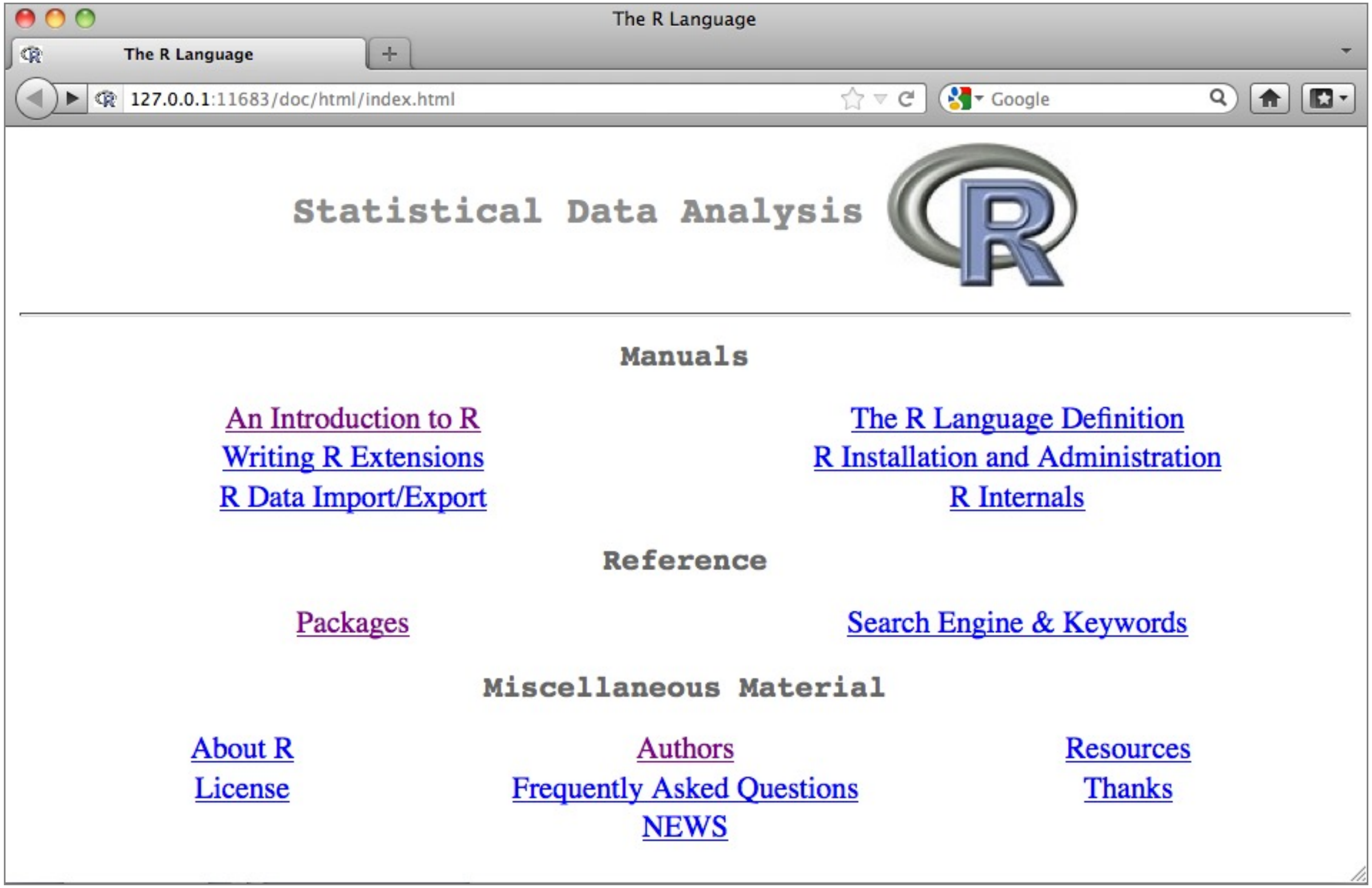
Viewing the Supplied Documentation

- **Problem**
  - You want to read the documentation supplied with R.

- **Solution**
  - Use the **help.start** function to see the documentation's table of contents.

```
> help.start()
```

Viewing the Supplied Documentation



Statistical Data Analysis

Manuals

An Introduction to R
Writing R Extensions
R Data Import/Export
The R Language Definition
R Installation and Administration
R Internals

Reference

Packages
Search Engine & Keywords

Miscellaneous Material

About R
License
Authors
Frequently Asked Questions
NEWS
Resources
Thanks

Getting Help on a Function

- **Problem**
  - You want to know more about a function that is installed.

- **Solution**
  - Use **help** to display the documentation for the function:

    > help(*functionname*)

    > ?*functionname*
  - Use **args** for a quick reminder of the function arguments:

    > args(*functionname*)
  - Use **example** to see examples of using the function:

    > example(*functionname*)

Searching the Supplied Documentation

- **Problem**
  - You want to search the installed documentation for a keyword.

- **Solution**
  - Use **help.search** to search the R documentation on your computer:

    > help.search("*pattern*")

    > ??*pattern*
  - You can see its documentation by explicitly telling help which package contains the function:

    > help(rma, package="affy")

Getting Help on a Package

- **Problem**
  - You want to learn more about a package installed.

- **Solution**
  - Use the **help** function and specify a package name:

    > help(package=*"packagename"*)
  - Use the **vignette** function to list all vignettes or vignettes for a certain package.

    > vignette()

    > vignette(package="*packagename*")
  - Use **vignette** function to view a certain vignette:

    > vignette("*vignettename*")

Searching the Web for Help

- **Problem**
  - You want to search the Web for information and answers regarding R.

- **Solution**
  - Use the **RSiteSearch** function to search by keyword or phrase:

    > RSiteSearch("*key phrase*")
  - Inside your browser, search the following sites:

http://rseek.org;

http://stackoverflow.com/

http://stats.stackexchange.com/

Finding Relevant Functions and Packages

- **Problem**
  - Of the 2,000+ packages for R, you have no idea which ones would be useful to you.

- **Solution**
  - Visit the list of task views at *http://cran.r-project.org/web/views/*.
  - Visit *http://rseek.org*, search by keyword.
  - Visit *http://crantastic.org/*, search by keyword.

Submitting Questions to the Mailing Lists

- **Problem**
  - You want to submit a question to the R community via the R-help mailing list.

- **Solution**
  - The Mailing Lists (*http://www.r-project.org/mail.html*) page contains general information and instructions for using the R-help mailing list.
  - The R mailing list should be your last choice. It is most likely that your question has already been answered, as very few questions are unique.

Getting and Setting the Working Directory

- **Problem**
  - You want to change your working directory. Or you just want to know what it is.
- **Solution**
  - Use **getwd** to report the working directory, and use **setwd** to change it:

```
> getwd()
[1] "/Users/yzhang”
> setwd("~/Teaching")
> getwd()
[1] "/Users/yzhang/Teaching"
```

  - Your working directory is the default location for all file input and output.

Saving Your Workspace

- **Problem**
  - You want to save your workspace without exiting from R.

- **Solution**
  - Call the **save.image** function:

    ```
    > save.image()
    ```

  - The workspace is written to a file called *.RData* in the working directory.

Viewing Your Command History

- **Problem**
  - You want to see your recent sequence of commands.
- **Solution**
  - Scroll backward by pressing the up arrow or Ctrl-P. Or use the **history** function to view your most recent input:

```
> history()
> history(100)
> history(Inf)
```

  - R saves the history in a file called *.Rhistory* in the working directory, if requested.

Saving the Result of the Previous Command

- **Problem**
  - You typed an expression into R that calculated the value, but you forgot to save the result in a variable.

- **Solution**
  - A special variable called **.Last.value** saves the value of the most recently evaluated expression:

```
> x <- .Last.value
```

Displaying the Search Path

- **Problem**
  - You want to see the list of packages currently loaded into R.

- **Solution**
  - Use the **search** function with no arguments:

    ```
    > search()
    ```
  - R uses the search path (in order) to find functions.
  - Your workspace (**.GlobalEnv**) is the first in the list. If your workspace and a package both contain a function with the same name, your workspace will “mask” the function.

Accessing the Functions in a Package

- **Problem**
  - When you try using functions in the package, however, R cannot find them. Why?
- **Solution**
  - Use either the **library** function or the **require** function to load the package into R:

    > library(*packagename*)

    > require(*packagename*)
  - The **detach** function will unload a package that is currently loaded:

    > detach(package:affy)

Accessing Built-in Datasets

- **Problem**
  - You want to use one of R’s built-in datasets.

- **Solution**
  - The standard **datasets** distributed with R are already available, since the datasets package is in search path:

    ```
    > data()                # Bring up a list of datasets
    ```

  - To access datasets in other packages, use the **data** function while giving the dataset name and package name:

    > data(*dsname*, package="*pkgname*")

Viewing the List of Installed Packages

- **Problem**
  - You want to know what packages are installed on your machine.
- **Solution**
  - Use the **library** function with no arguments for a basic list. Use **installed.packages** to see more detailed information about the packages:

```
> library()
> installed.packages()[,c("Package","Version")]
```

Installing Packages from CRAN

- **Problem**
  - You found a package on CRAN, and now you want to install it on your computer.
- **Solution**
  - Use the **install.packages** function:

    > install.packages("*packagename*")
  - On Linux or Unix systems, root privileges are required to install packages into the system-wide libraries.
  - If the new package depends upon other packages that are not already installed locally, then the R installer will automatically download and install those required packages.

Suppressing the Startup Message

- **Problem**
  - You are tired of seeing R’s verbose startup message.

- **Solution**
  - Use the **--quiet** command-line option when you start R:

```
R --quiet
```

Running a Script

- **Problem**
  - You captured a series of R commands in a text file. Now you want to execute them.

- **Solution**
  - The **source** function instructs R to read the text file and execute its contents:

    ```
    > source("myScript.R”)
    ```
  - Setting **echo=TRUE** will echo the script lines before they are executed:

    ```
    > source("hello.R", echo=TRUE)
    ```

Running a Batch Script

- **Problem**
  - You want to execute an R script.
- **Solution**
  - Run the R program with the **CMD BATCH** subcommand:

    R CMD BATCH *scriptfile outputfile*
  - The **CMD BATCH** subcommand normally calls **proc.time** when your script completes. To prevent calling **proc.time**, end your script by calling **q** function with **runLast=FALSE**:

    ```
    > q(runLast=FALSE)
    ```
  - Other useful options in batch mode:

    --slave; --no-restore; --no-save; --no-init-file

Running a Batch Script

- **Solution**
  - To pass command-line arguments to the script, use the **Rscript** command:

    ```
    Rscript scriptfile arg1 arg2 arg3
    ```
  - Inside the script, the command-line arguments can be accessed by calling **commandArgs**

    ```
    argv <- commandArgs(TRUE)
    ```
  - In Linux or Unix, place a #! line at the head with the path to the **Rscript** program:

    ```
    #!/usr/bin/Rscript --slave
    argv <- commandArgs(TRUE)
    x <- as.numeric(argv[1])
    ```

Printing Something

- **Problem**
  - You want to display the value of a variable or expression.

- **Solution**
  - Enter the variable name or expression at the command prompt, R will print its value:

```
> pi
[1] 3.141593
```

  - Use **print** function for generic printing of any object:

```
> print(pi)
[1] 3.141593
```

Printing Something

- The **print** function prints only one object at a time:

```
> print("The zero occurs at", 2*pi, "radians.")        # Wrong!
> print("The zero occurs at"); print(2*pi); print("radians") # Right!
```

- The **cat** function can print multiple items, which puts a space between each item by default. A newline character (\n) should be provided to terminate the line:

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> cat("The first few Fibonacci numbers are:", fib, "...\n")
```

- The **cat** function cannot print compound data structures such as matrices and lists:

```
> cat(list("a","b","c"))      # Wrong!
```

Setting Variables

- **Problem**
  - You want to save a value in a variable.
- **Solution**
  - Use the assignment operator (<-):

    ```
    > x <- 3
    ```

  - R is a *dynamically typed language*, which means that we can change a variable’s data type at will:

    ```
    > x <- 3
    > x <- c("fee", "fie", "foe", "fum")
    ```

Listing Variables

- **Problem**
  - To list variables and functions defined the workspace.

- **Solution**
  - Use the **ls** function. Use **ls.str** for more details (both listing variables and applying the **str** function to show their structure):

```
> ls()
[1] "fib" "x"
> ls.str()
fib :  num [1:10] 0 1 1 2 3 5 8 13 21 34
x :  chr [1:4] "fee" "fie" "foe" "fum"
```

Deleting Variables

- **Problem**
  - You want to remove unneeded variables or functions from your workspace or to erase its contents completely.

- **Solution**
  - Use the **rm** function:

```
> rm(x, fib)
> rm(list=ls())  # To erase your entire workspace at once.
```

  - Never put rm(list=ls()) into code you share with others, such as a library function or sample code sent to a mailing list.

Creating a Vector

- **Problem**
  - You want to create a vector.
- **Solution**
  - Use the **c(...)** operator to construct a vector:

```
> c(1,1,2,3,5,8,13,21)
> c("Everyone", "loves", "stats.”)
> c(TRUE,TRUE,FALSE,TRUE)
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> c(v1,v2)
[1] 1 2 3 4 5 6
```

Creating a Vector

– Vectors cannot contain a mix of data types:

```
> v1 <- c(1,2,3)
> v3 <- c("A","B","C”)
> c(v1,v3)
[1] "1" "2" "3" "A" "B" "C”
```

– Two data elements can coexist in a vector only if they have the same mode:

```
> mode(3.1415)
[1] "numeric"
> mode("foo")
[1] "character"
> mode(c(3.1415, "foo"))
[1] "character"
```

Computing Basic Statistics

- **Problem**
  - You want to calculate basic statistics.
- **Solution**
  - Use the simple functions as follows:

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> y <- log(x+1)
> mean(x)
> median(x)
> sd(x)
> var(x)
> cor(x,y)
> cov(x,y)
```

Computing Basic Statistics

– Values that are not available:

```
> x <- c(0,1,1,2,3,NA)
> mean(x)
[1] NA
```

– If necessary, you can tells R to ignore the NA values:

```
> mean(x, na.rm=TRUE)
[1] 1.4
```

Computing Basic Statistics

– **var**, **cor** and **cov** function on data frame:

```
> cor(cars)
          speed      dist
speed 1.0000000 0.8068949
dist  0.8068949 1.0000000
> cov(cars)
          speed     dist
speed  27.95918 109.9469
dist  109.94694  664.0608
> var(cars)
          speed     dist
speed  27.95918 109.9469
dist  109.94694 664.0608
```

Creating Sequences

- **Problem**
  - You want to create a sequence of numbers.
- **Solution**
  - Use an **n:m** expression to create the simple sequence:

```
> 10:15
[1] 10 11 12 13 14 15
> 15:10
[1] 15 14 13 12 11 10
```

  - Use the **seq** function for sequences with an increment other than 1:

```
> seq(from=0, to=20, by=2)
[1]  0  2  4  6  8 10 12 14 16 18 20
```

Creating Sequences

```
> seq(from=20, to=0, by=-2)
 [1] 20 18 16 14 12 10  8  6  4  2  0
> seq(from=0, to=2, length.out=5)
[1] 0.0 0.5 1.0 1.5 2.0
```

– Use the **rep** function to create a series of repeated values:

```
> rep(1, times=5)
[1] 1 1 1 1 1
```

Comparing Vectors

- **Problem**
  - You want to compare two vectors or you want to compare an entire vector against a scalar.
- **Solution**
  - The comparison operators (**==, !=, <, >, <=, >=**) can perform comparison:

```
> v <- c(3, pi, 4)
> w <- c(pi, pi, pi)
> v == w
[1] FALSE  TRUE FALSE
> v < w
[1]  TRUE FALSE FALSE
```

Comparing Vectors

- Compare a vector against a single scalar:

```
> v == pi
[1] FALSE  TRUE FALSE
> v < pi
[1]  TRUE FALSE FALSE
```

- The **any** and **all** functions:

```
> any(v == pi)
[1] TRUE
> all(v == pi)
[1] FALSE
```

Selecting Vector Elements

- **Problem**
  - You want to extract one or more elements from a vector.
- **Solution**
  - Use square brackets **[]** to select vector elements by their position.
  - Use negative indexes to exclude elements.
  - Use a vector of indexes to select multiple values.
  - Use a logical vector to select elements based on a condition.
  - Use names to access named elements.

Selecting Vector Elements

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> fib[1]              # The first element has an index of 1.
[1] 0
> fib[4:9]
[1]  2  3  5  8 13 21
> fib[-1]             # Ignore first element.
[1]  1  1  2  3  5  8 13 21 34
> fib[c(1,2,4,8)]
[1]  0  1  2 13
> fib[fib > 10]
[1] 13 21 34
> fib[fib %% 2 == 0]
[1]  0  2  8 34
```

Selecting Vector Elements

– Select all elements greater than the median.

```
> v[ v > median(v) ]
```

– Select all elements in the lower and upper 5%.

```
> v[ (v < quantile(v,0.05)) | (v > quantile(v,0.95)) ]
```

– Select all elements that exceed 2 standard deviations from the mean.

```
> v[ abs(v-mean(v)) > 2*sd(v) ]
```

– Select all elements that are neither NA nor NULL

```
> v[ !is.na(v) & !is.null(v) ]
```

Selecting Vector Elements

```
> years <- c(1960, 1964, 1976, 1994)
> names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")
> years
Kennedy Johnson  Carter Clinton
1960    1964    1976    1994
> years[c("Carter", "Clinton")]
 Carter Clinton
1976    1994
```

Performing Vector Arithmetic

- **Problem**
  - You want to operate on an entire vector at once.

- **Solution**
  - The usual arithmetic operators can perform element-wise operations on entire vectors:

```
> v <- c(11,12,13,14,15)
> w <- c(1,2,3,4,5)
> v + w
[1] 12 14 16 18 20
> w ^ v
[1]   1        4096     1594323   268435456 30517578125
```

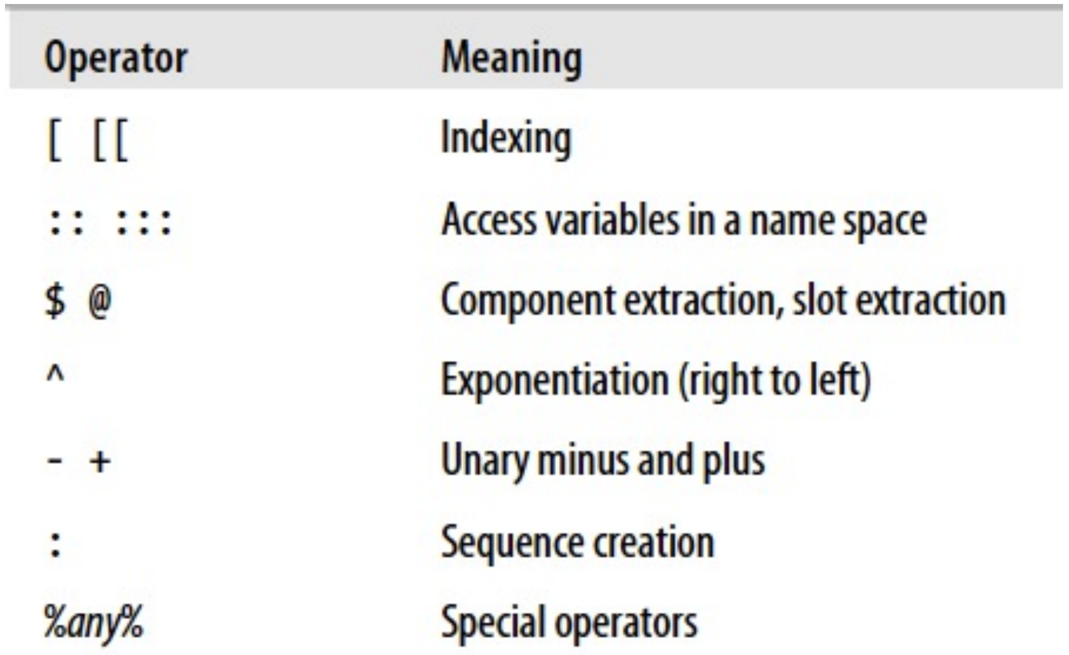
Performing Vector Arithmetic

```
> w + 2
[1] 3 4 5 6 7
> w ^ 2
[1]  1  4  9 16 25
> w - mean(w)
[1] -2 -1  0  1  2
> (w - mean(w)) / sd(w)
[1] -1.2649111 -0.6324555  0.0000000  0.6324555  1.2649111
> log(w)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> sqrt(w)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> sin(w)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
```

Getting Operator Precedence Right

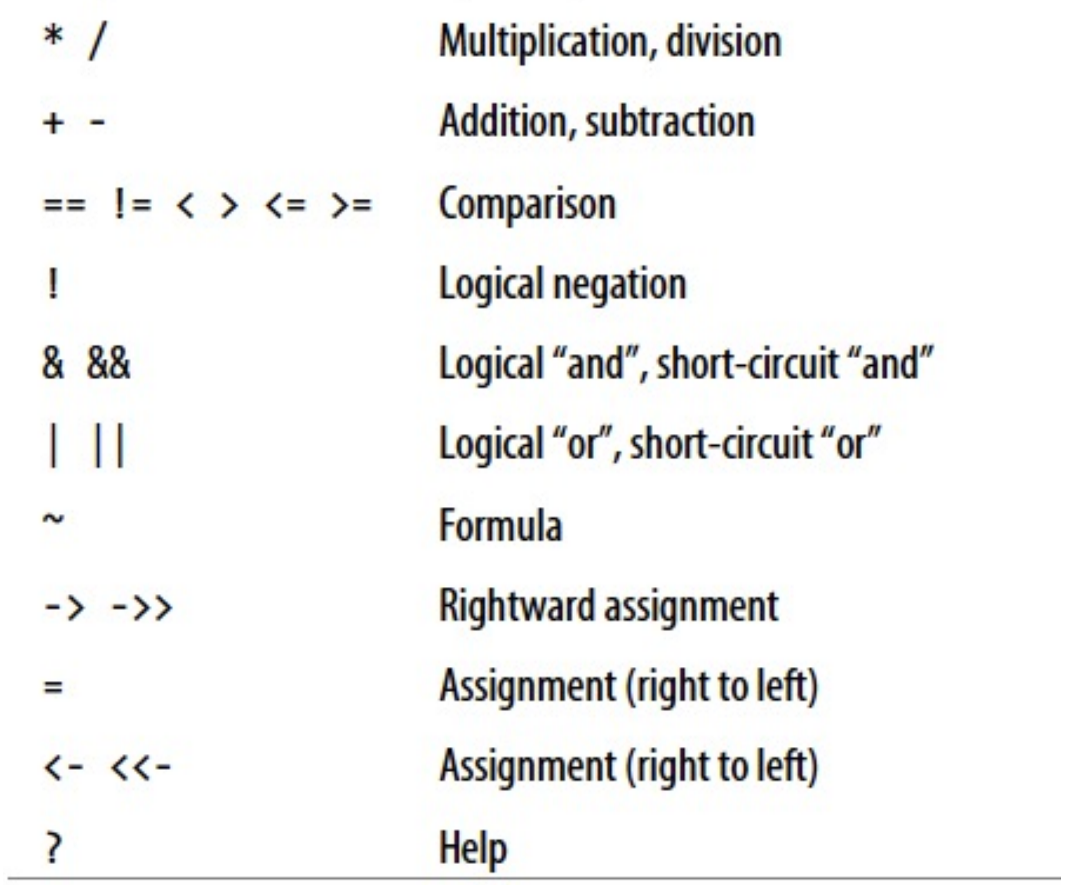
- **Problem**
  - If operator precedence is the cause for unexpected results.

- **Solution**
  - The operator precedence is as follows:



Operator	Meaning
`[ [[`	Indexing
`:: :::`	Access variables in a name space
`$ @`	Component extraction, slot extraction
`^`	Exponentiation (right to left)
`- +`	Unary minus and plus
`:`	Sequence creation
%any%	Special operators

Getting Operator Precedence Right



`* /`	Multiplication, division
`+ -`	Addition, subtraction
`== != < > <= >=`	Comparison
`!`	Logical negation
`& &&`	Logical “and”, short-circuit “and”
`| ||`	Logical “or”, short-circuit “or”
`~`	Formula
`-> ->>`	Rightward assignment
`=`	Assignment (right to left)
`<- <<-`	Assignment (right to left)
`?`	Help

```
> n <- 10
> 0:n-1              # identical to (0:n)-1
[1] -1  0  1  2  3  4  5  6  7  8  9
```

Defining a Function

- **Problem**
  - You want to define an R function.

- **Solution**
  - Create a function by using the **function** keyword followed by a list of parameters and the function body:

    function(*param*$_1$, ..., *param*$_N$) *expr*

    function(*param*$_1$, ..., *param*$_N$) {

    expr$_1$

    …

    expr$_M$

    }

Defining a Function

```
> cv <- function(x) sd(x)/mean(x)              # coefficient of variation
> cv(1:10)
[1] 0.5504819

> gcd <- function(a,b) {    # compute the greatest common divisor
+ if (b == 0) return(a)
+ else return(gcd(b, a %% b))
+ }
> gcd(25, 15)
[1] 5
```

Input and Output

- R is not a great tool for preprocessing data files.

- If your data is difficult to access or difficult to parse, consider using an outboard tool (python, perl, awk, etc) to preprocess the data before loading it into R. Let R do what R does best.

Printing Fewer Digits or More Digits

- **Problem**
  - R normally formats floating-point output to have seven digits, but you want more or fewer digits.
- **Solution**
  - Use the **digits** parameter in **print** function, or use the **format** function (also has **digits** parameter) in **cat** function:

```
> print(pi, digits=4)
[1] 3.142
> cat(format(pi,digits=4), "\n")
3.142
> print(pnorm(-3:3), digits=3)
[1] 0.00135 0.02275 0.15866 0.50000 0.84134 0.97725 0.99865
```

Printing Fewer Digits or More Digits

```
> q <- seq(from=0,to=3,by=0.5)
> tbl <- data.frame(Quant=q, Lower=pnorm(-q), Upper=pnorm(q))
> print(tbl,digits=2)
  Quant  Lower Upper
1   0.0 0.5000  0.50
2   0.5 0.3085  0.69
3   1.0 0.1587  0.84
4   1.5 0.0668  0.93
5   2.0 0.0228  0.98
6   2.5 0.0062  0.99
7   3.0 0.0013  1.00
```

Redirecting Output to a File

- **Problem**
  - You want to redirect the output from R into a file.

- **Solution**
  - Use the **file** argument in **cat** function:

```
> cat(v, "\n", file="output.txt")
> cat(w, "\n", file="output.txt", append=TRUE)

> con <- file("output-2.txt", "w")
> cat(v, "\n", file=con)
> cat(w, "\n", file=con)
> close(con)
```

Redirecting Output to a File

– Use the **sink** function to redirect *all* output:

```
> sink("script_output.txt")
> source("hello.R")
> sink()
```

Listing Files

- **Problem**
  - You want to see a listing of files in working directory.

- **Solution**
  - The **list.files** function shows the contents:

```
> list.files()
> list.files(recursive=T)
> list.files(all.files=TRUE)
```

Reading Fixed-Width Records

- **Problem**
  - You are reading data from a file of fixed-width records.
- **Solution**
  - Read the file using the **read.fwf** function:

```
> records <- read.fwf("fixed-width.txt", widths=c(10,10,4,-1,4),
+             col.names=c("Last","First","Born","Died"))
> records
         Last      First Born Died
1 Fisher     R.A.       1890 1962
2 Pearson    Karl       1857 1936
3 Cox        Gertrude   1900 1978
4 Yates      Frank      1902 1994
5 Smith      Kirstine   1878 1939
```

Reading Tabular Data Files

- **Problem**
  - You want to read a text file that contains a table of data.
- **Solution**
  - Use the **read.table** function, which returns a data frame.
  - By default, field separator is white space (blanks or tabs); it can be specified by **sep** parameter.
  - To prevent **read.table** from interpreting character strings as factors, set the **stringsAsFactors** parameter to FALSE.

```
> dfrm <- read.table("statisticians.txt", header=TRUE,
+          stringsAsFactor=FALSE)
```

Reading from CSV Files

- **Problem**
  - You want to read data from a comma-separated values (CSV) file.

- **Solution**
  - Use **read.csv** function for CSV files with a header line, set **header** option to FALSE for CSV files w/o header line.
  - The **read.csv** function will not interpret nonnumeric data as a factor, if set the **as.is** parameter to TRUE.

```
> tbl <- read.csv("table-data.csv", as.is=TRUE)
```

Writing to CSV Files

- **Problem**
  - You want to save a matrix or data frame in a file using the comma-separated values format.

- **Solution**
  - The **write.csv** function can write a CSV file:

```
> write.csv(tbl, file="table-data_new.csv", row.names=FALSE)
```

Reading Data from HTML Tables

- **Problem**
  - You want to read data from an HTML table on the Web.

- **Solution**
  - Use the **readHTMLTable** function in the **XML** package:

```
> library(XML)
> url <- 'http://www.stat-nba.com/team/BOS.html'
> tbl <- readHTMLTable(url, which=1)
```

Reading from MySQL Databases

- **Problem**
  - You want access to data stored in a MySQL database.
- **Solution**
  - Use functions in **RMySQL** package:

```
> library(RMySQL)
> con = dbConnect(MySQL(), user="genomep",
+          password="password", dbname="hg19",
+          host="genome-mysql.cse.ucsc.edu")
> dbListTables(con)
> sql <- "SELECT * from refGene"
> refGene <- dbGetQuery(con, sql)
> dbDisconnect(con)     # close the database connection
```

Saving and Transporting Objects

- **Problem**
  - You want to store R objects in a file for later use.

- **Solution**
  - Write the objects to a file using the **save** function, Read them back using the **load** function:

```
> save(refGene, file="refGene.RData")
> load("refGene.RData")
```

Summary

- R Cookbook
  - Chapter 1. Getting Started and Getting Help
  - Chapter 2. Some Basics
  - Chapter 3. Navigating the Software
  - Chapter 4. Input and Output