任何人可以访问互联网，几乎可以在任何计算机上安装 R。可以访问 cran.r-project.org 并获得安装程序或源代码。二进制发行版（例如，预编译的 Windows、Debian 等）需要最小的基础结构。源代码发行版需要编译器。

当 R 安装后，使用 `install.packages` 或 GUI 来安装 CRAN 上的包。可以使用 `source("http://www.bioconductor.org/getBioC.R")`、`getBioC()` 或 `reposTools::install.packages` 来安装 Bioconductor 上的包。

注意 R 版本和包版本的一致性。

### 你将会获得
- 交互式编程语言
- 可脚本化的编程语言（存储程序源代码）
- 图形用户界面（全点-点击界面，如 affylmGUI）
- 可视化支持，具有一定的交互性，可在多种常见图形格式中渲染
- 包协议用于软件共享、组织软件、示例或研究数据、文档、软件质量控制
- 门户（CRAN、bioconductor、邮件列表）用于构建相互作用的用户和开发者社区
R in a nutshell: getting help

- obtain R and installation instructions from www.r-project.org
- help.start() fires up a browser-based interface
- help.search("[keyword]") scour all available manual pages for titles or concepts including the string [keyword]
- help([topic]) finds specific manual information on the named topic
- mail to R-help@stat.math.ethz.ch gets access to a lively community of users and developers

Different ways of using R

- all platforms: command line interface (Rterm on windows)
- Windows and Mac: GUIs with buttons for package management
- browser-based interfaces exist
- can fit into pipelines, so users are not directly exposed to R language

Rigor

- using R interactively, you must attend to rules of syntax interactively
- all quote-marks, parentheses and brackets must balance, statements must be terminated either by linefeeds or semicolons
- to cause a function to execute, you must supply arguments in parentheses:
  \[
  \text{sqrt(2)}
  \]
  \[
  \begin{align*}
  \text{sqrt(2)} & \\
  & \text{[1]} \ 1.414214
  \end{align*}
  \]
- for beginners, errors of misspelling symbol or command names, and of failing to balance parentheses or quotation marks, are very common
- emacs users can use ESS, which includes symbol completion facilities

Input continuation

- interactive use can involve typing long sequences of parameters or multi-line function definitions
- R will prompt with > at the inception of an interaction and with + when starting a new line within a command
- if you see a + prompt unexpectedly, you have probably left a parenthesis or quotation mark unmatched.
- the control-C or stop-sign interrupt can typically be used
Bad things happen

- you can crash R by running bugged software or by exceeding machine resources or by ...
- when you have a valuable series of interactive computations, use `savehistory` and/or `save.image` so that a crash won’t force you to recreate
- when using a GUI, the system control can go to non-dialogue panels; check this before getting too worried about a crash

what happened?

- we see that there are thousands of objects comprising the various packages loaded when R is begun
- we loaded the `affy` package using `library`
- a number of packages that `affy` depends upon (such as `Biobase`) were loaded to support `affy`
- the total number of objects in scope is very large
- you do not need to know them all by name, but the more you know, the better

a view of the base

```r
> sapply(search(), function(x) length(objects(x, all = TRUE)))

.GlobalEnv    package:stats    package:graphics    package:grDevices
           16             487              84              62
package:utils    package:datasets    package:methods    package:lattice
           133            103             291             104
Autoloads    package:base
           1             1075
```

```r
> library(affy)
> sapply(search(), function(x) length(objects(x, all = TRUE)))

           16             166             271             143
package:tools    package:stats    package:graphics    package:grDevices
           40             487              84              62
package:utils    package:datasets    package:methods    package:lattice
           133            103             291             104
Autoloads    package:base
           1             1075
```

R in a nutshell: persistence, globals

- persistence: save workspace when closing application
- when you start R again, all objects created and not removed in previous session are restored
- `getwd()` tells current working disk directory
- `setwd([path])` modifies working disk directory
- `savehistory()` saves audit of computations
- `save.image()` saves all objects currently in use
- loaded libraries must be manually reloaded
- global options list manipulated by `options`
- global graphical options by `par`
R in a nutshell: basic data structures

- organizing information for programming and analysis
  - atoms: either numeric or character
  - vector: linear structure either numeric or character, arbitrary length
  - factor: representation of categorical information
  - matrix: two-dimensional structure, either numeric or character; array generalizes matrix
  - list: linear structure, arbitrary element modes
  - data.frame: two-dimensional structure, viewable as mix of numeric and character

Symbol resolution, workspace

- assignment of a value to a symbol is accomplished by `<-` or `=`
- a workspace called `.GlobalEnv` is the dynamic repository of all creations in a session
- persistent symbols can be identified using `objects()`
- the entire search space for symbol resolution is identified using `search()`
- objects in `.GlobalEnv` are removed using `rm()`; searchlist elements are dropped using `detach()`

R: names attributes; "associative" indexing

- numeric indexing of linear structures is fine
- names of elements are also useful

```r
> x <- c(1, 2, 3, 4)
> x[2]
[1] 2
> y <- c(5, 6, 7, 8)
> y[c(2, 3)]
[1] 6 7
> m <- cbind(x, y)
> m
     x y
[1,] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
> m[, "x"]
 [1] 1 2 3 4
```

```r
> names(x) <- c("a", "b", "c", "d")
> x["b"]
b2
> rownames(m) <- LETTERS[1:4]
> m
     x y
  A 1 5
  B 2 6
  C 3 7
  D 4 8
> m["A", "y"]
[1] 5
```
upshots of associative indexing

```r
> litdf <- data.frame(samp1 = c(33, 22, 12), samp2 = c(44, 111, 13))
>rownames(litdf) <- c("CRP", "BRCA1", "HOXA")
> litdf

 samp1 samp2
CRP 33 44  
BRCA1 22 111
HOXA 12 13
> litdf["CRP",]

 samp1 samp2
CRP 33 44  
> litdf[, "samp1"]

[1] 33 22 12
> litdf["HOXA", "samp2"]

[1] 13
```

Selection rules

- explicit numeric selection
- named element selection
- logical filtering

```r
> x

a b c d
1 2 3 4
> y

[1] 5 6 7 8
> keep <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
> x[keep]

a b <NA> 1 2
> x[y > 6]

c d
3 4
> which(y > 6)

[1] 3 4
```

Other containers

```r
> gender <- factor(c("M", "M", "F", "F"))
> gender

[1] M M F F
Levels: F M
> season <- ordered(c("spring", "summer", "fall", "winter"), levels = c("spring", + "summer", "fall", "winter"))
> season

[1] spring summer fall winter
Levels: spring < summer < fall < winter
> df

x y gender season
A 1 5 M spring
B 2 6 M summer
C 3 7 F fall
D 4 8 F winter
> df$gender

[1] M M F F
Levels: F M
> df["B",]

x y gender season
B 2 6 M summer
> x <- "season"
> df[[x]]

[1] spring summer fall winter
Levels: spring < summer < fall < winter
```

Accessing data.frame elements

```r
> df

x y gender season
A 1 5 M spring
B 2 6 M summer
C 3 7 F fall
D 4 8 F winter
```

VJ Carey  |  bioc computing
Merging data.frames

> df
  x  y gender season
  A 1 5 M spring
  B 2 6 M summer
  C 3 7 F fall
  D 4 8 F winter

> df2
  x  z
  1 2 1
  2 3 6
  3 4 7
  4 5 8

> merge(df, df2)
  x  y gender season z
  1 2 6 M summer 12 3 7 F fall 63 4 8 F winter 7

finds matching attributes and concatenates

Conservative merge

> merge(df, df2, all = TRUE)
  x  y gender season z
  1 1 5 M spring NA
  2 2 6 M summer 1
  3 3 7 F fall 6
  4 4 8 F winter 7
  5 5 NA <NA> <NA> 8

other parameters for alternate combos

lists

> l1 <- list(df, x = x, fundem = mean)
> l1

[[1]]
  x  y gender season
  A 1 5 M spring
  B 2 6 M summer
  C 3 7 F fall
  D 4 8 F winter

$z
a b c d
1 2 3 4
$fundem
function (x, ...)
UseMethod("mean")
<environment: namespace:base>

split vectors to make lists

> dx <- c("ALL", "ALL", "AML", "AML", "ALL", "ALL", "ALL", "AML")
> ddr1 <- c(12.2, 13.1, 7.2, 6.4, 14.2, 15.3, 9.2, 10)
> split(ddr1, dx)

$ALL
[1] 12.2 13.1 7.2 6.4
$AML
[1] 14.2 15.3 9.2 10
access to list elements

```r
> l1[[1]]
x  y  gender  season
A 1  5  M  spring
B 2  6  M  summer
C 7  3  F  fall
D 8  4  F  winter
```

```r
> l1$x
a b c d
1 2 3 4
```

```r
> l1$gender

function (x, ...) UseMethod("mean")<environment: namespace:base>
```

Single and double bracket selection

- subset vectors using single brackets
  - `x[2:3]` is a 2-vector
- subset lists to sublists using single brackets
  - `11[1:2]` is a list of 2 elements
- retrieve list elements using double brackets or `$`
  - `11[[2]]` is the second element of `11`

iterated access

```r
> for (i in 1:2) print(i, season[i])

[1] 1
[1] 2
```

```r
> for (i in 1:3) print(l1[i])

[[1]]
x  y  gender  season
A 1  5  M  spring
B 2  6  M  summer
C 7  3  F  fall
D 8  4  F  winter
```

random data

```r
> table(rpois(1000, 3))
0  1  2  3  4  5  6  7  8  9
51 149 219 228 166 116 38 29 3 1
```

```r
> table(rbinom(1000, 5, 0.5))
0  1  2  3  4  5
26 144 281 351 158 40
```

```r
> sd(rnorm(1000, 0, 0.4))
[1] 0.4059641
```
data simulation and recovery of parameters

```r
> set.seed(1234)
> x <- rnorm(3000, 12, 2)
> mean(x)
[1] 12.01124
> sd(x)
[1] 1.993966
> y <- rpois(3000, 4.7)
> mean(y)
[1] 4.697
> var(y)
[1] 4.50736
```

VJ Carey bioc computing

CIs, tests

```r
t.test(x)
One Sample t-test
data: x
t = 329.9369, df = 2999, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
11.93986 12.08262
sample estimates:
mean of x
12.01124

wilcox.test(x, y)
Wilcoxon rank sum test with continuity correction
data: x and y
W = 8924708, p-value < 2.2e-16
alternative hypothesis: true mu is not equal to 0

cor.test(x, y)
Pearson's product-moment correlation
data: x and y
t = -1.6432, df = 2998, p-value = 0.1005
alternative hypothesis: true correlation is not equal to 0

cor
-0.02999639
```

VJ Carey bioc computing

dose-response model

- `x` is a series of doses
- `y` is the count of deaths with dose `x`

```r
dose <- c(-1,199, 199,400, 400,600, 600,801, 801,1e+03)
dev <- c(0, 57, 42, 19, 2, 11, 3, 18, 44, 58, 59)
dev <- dev/10

summary(glm <- glm(dev ~ dose, fam = binomial))

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
 (Intercept)  9.358388  0.481256  19.468 2.26e-16 ***
dose 0.0090921 0.0011009  8.278 0.000166 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Dispersion parameter for binomial family taken to be 1

Null deviance: 407.68 on 302 degrees of freedom
Residual deviance: 191.13 on 301 degrees of freedom
AIC: 195.13

Number of Fisher Scoring iterations: 6
```

VJ Carey bioc computing

logistic fit

```r
summary(gl <- glm(res ~ do, fam = binomial))

Deviance Residuals:
Min 1Q Median 3Q Max
-2.9032 -0.4172 0.1163 0.4044 2.6349

Coefficients:
 Estimate Std. Error t value Pr(>|t|)
 (Intercept)  -3.935838  0.481256  -8.178 2.88e-16 ***
dose 0.009921  0.001100   9.019 < 2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 407.68 on 302 degrees of freedom
Residual deviance: 191.13 on 301 degrees of freedom
AIC: 195.13

Number of Fisher Scoring iterations: 6
```
display

with fitted LD50

simple visualization

> hist(rnorm(1000))
> abline(v = 0, col = "red", lwd = 3)

histogram of rnorm(1000)

> hist(rnorm(1000))
> abline(v = 0, col = "red", lwd = 3)

Histogram of rnorm(1000)

Visualization principles

- use high-level procedures when available (save formatting decisions and tweaking)
- get acquainted with data visualization guidelines (Tuft, Visualization of Quantitative Information, Wilkinson Grammar of graphics)
- for one-offs, learn about `par`, `plot`, `points`, `text`, `legend`, ...
- most plots built by a series of commands
  - initial layout, exterior annotation (title, axes)
  - enhancements
  - interior annotation (legend, highlights)
- for designing visualization tools, learn about `grid`
alterations: label orientation

\begin{verbatim}
after par(las=2)
\end{verbatim}

\begin{verbatim}
plot(x,y,log='x')
\end{verbatim}

alterations: log (log-log also avail.)

\begin{verbatim}
log (log-log also avail.)
\end{verbatim}

\begin{verbatim}
plot(x,log='x')
\end{verbatim}

\begin{verbatim}
plot(x,y,log='x')
\end{verbatim}
simple scatterplot matrix

Edgar Anderson's Iris Data (pairs(iris[1:4]))

enhanced scatterplot matrix

bg = c('red', 'green3', 'blue')[unclass(iris$Species)]

challenge

- Previous plot is nice but has no key
- Where would you put a key?
- How would you do it? Consider locator() to get coordinates from mouse-clicks, or alternate design of the plot?

contours

A Topographic Map of Maunga Whau

Meters North

Meters West
functions

- a function has a parameter list and a body
- the parameter lists identify inputs
- the bodies manipulate inputs and other entities to compute and return a final value
- symbol resolution occurs by inspecting symbols defined in the current function body, its enclosing environments, and then traversing the searchlist

functions

- x and y are formal parameters
- + and * are primitive functions
- arithmetic combination of vector and scalar is permitted
- binary operator functions obey recycling rule
Recycling rule

- when a binary operator has operands of different lengths, the shorter is replicated to the length of the longer (with truncation if necessary)
- the binary operation is then executed elementwise
- warning if replication leads to truncation

```r
> 4 * c(2, 3, 4, 5)
[1] 8 12 16 20
> c(2, 3) * c(4, 5, 6)
[1] 8 15 12
```

Defining functions

- you can define them interactively on the fly as above
- you can write a text file containing the definition and use `source` to load that into `.GlobalEnv` for any session
- you can put the function in the R directory of a source package and then INSTALL the package (after documenting the function)
- you CAN NOT add a function to an INSTALLed package, don’t even try

Applying functions

- it is often of interest to apply a function repetitively over all the rows or columns of a matrix, or all the elements of a list or vector

```r
> x
A 1 5
B 2 6
C 3 7
D 4 8
> apply(x, 1, sum)
A 6
B 8
C 10
D 12
```

apply family

- `apply(x, MARGIN, FUN, ...)`, `x` is a matrix or array, `MARGIN` is 1 for rows, 2 for columns, etc., `FUN` is a function to be evaluated on the extracted vectors; the dots are there to accept additional fixed arguments to be supplied to `FUN`
- `lapply(x, FUN, ...)` returns a list of values returned by `FUN` applied to elements of list `x`
- `sapply(x, FUN, ...)` returns the simplest sensible structure of values returned by `FUN` applied to elements of list `x`
- `mapply(FUN, ..., MoreArgs)` is an apply of multiparameter functions over multiple vectors
Summary thus far

- session concepts: `.GlobalEnv`, searchlist, history, save.image, savehistory
- basic containers: vector, matrix, array, list, data.frame
- basic constructor functions: `c`, `cbind`, `rbind`, `matrix`, `data.frame`
- basic vector or matrix element access: `[]` with numeric arguments, or `[]` with name strings
- basic list or data.frame component access: `$` or `[[]`
- functions can be constructed on the fly or in script files
- functions can be repetitively applied with the `apply` family

Quiz questions

- what is
  > `c(2, 3, 4, 5, 6)[1:5%%2 == 1]`

Quiz questions

- what is
  > `matrix(c(2, 3, 4), nr = 3, nc = 2)`

Quiz questions

- Let
  > `myl <- list(sin, cos, tan)`
  What is
  > `sapply(myl, function(f) f(pi/4)^2)`
Let

```
> sbp <- c(122, 113, 118, 204, 117, 108, 122, 104, 97)
> state <- c(rep("NY", 5), rep("NJ", 4))
> age <- c(23, 43, 22, 53, 32, 44, 22, 55, 21)
```

What are

```
> split(sbp, gend)
> sapply(split(sbp, gend), mean)
```

---

**Packages**

- packages are collections of data, functions, datastructure definitions, documentation in a regimented system of folders
- CRAN and Bioconductor are major repositories of packages
- package installation is governed by R (use the GUI or install.packages family)
- package access is obtained either manually (library calls) or automatically (by package interdependency resolution)

---

**Summary**

- R is vector-oriented
- vector or matrix elements can be referred to by location or name
- logic, arithmetic, iteration have natural expressions
- many approaches to customizable visualization
- functional program units: `f <- function(x,y)body`
- built-in functions do data shaping, statistics, visualization
- packages (and functions you write) extend R’s built-in functions (temporarily)

---

**a stroll through CRAN ...**

- applied statistics, modern regression: MASS, gam, boot, mgcv
- machine learning support: e1071, cluster, som
- visualization: scatterplot3d, rgl, lattice
- inference with dependent data: nlme, geepack, tseries, dse
- phylogenetics: ape
- ... ['task views' to organize resources are under development]