

Методические указания по дисциплине

Разработка интеллектуальных систем на языке R

1.1. Установка, запуск и элементарные команды для среды R. Работа в консоли и в графической оболочке

Задание

Установить и запустить R в режиме терминала и GUI. Запустить все команды из разделов 1.1.1 и 1.1.2.

1.1.1. Установка и запуск R

R можно загрузить с сайта <http://cran.r-project.org/>. Предварительно скомпилированные загрузочные файлы доступны для Linux, Mac OS X, Windows. Функциональность программы можно расширять с помощью пакетов, они также доступны с сайта CRAN (Comprehensive R Archive Network). У этого сайта довольно много зеркал (копий), так что выбирайте подходящее. Скачать можно как исходный код для последующей компиляции, так и собранный бинарный пакет (последнее, не всегда возможно). Исходный код R компилируется как правило без проблем.

Linux. Если у вас какой-нибудь из распространенных дистрибутивов, то R наверняка входит в репозиторий прилагающихся к вашей системе пакетов. Единственное, что нужно учесть, - обновление пакетов часто отстает от выхода версий R (как правило, четыре раза в год).

Версия нумеруется тремя числами, первые два - это главная версия, обновляется два раза в год. С каждой главной версией в R вносятся изменения, часто довольно значительные. Как правило, это множество новых команд, исправленные алгоритмы выполнения старых, и разумеется, исправления ошибок. Кроме того, бывает так, что страдает обратная совместимость - новые версии начинают иначе истолковывать команды, а иногда команды или их опции перестают действовать. Естественно, разработчики стараются минимизировать такие изменения. В общем, рациональный подход такой: обновляйте R, но при этом всегда читайте список изменений версии. На каждую главную версию выходят, как правило, две минорные версии (нулевая и первая). Первая минорная версия обычно ничего нового, кроме исправления ошибок, не содержит.

Mac OS X. Для того чтобы начать работать с R в Mac OS X, надо сначала убедиться, что у вас последняя версия этой операционной системы.

Последние версии R выходят только под последние версии Mac OS. Установочный пакет скачивается с CRAN (не забудьте также скачать оттуда Tcl/Tk). Имейте в виду, что графическая оболочка R для Mac (R Mac GUI) обновляется быстрее, чем сам R, поэтому разработчики предусмотрели возможность скачивания и установки оболочки отдельно. Установка происходит практически так же, как установка любой программы под Mac. Надо еще отметить, что R завоевал себе популярность не в последнюю очередь тем, что он запускался под Mac, в то время как S-Plus под эту платформу был недоступен.

Windows. Для того чтобы запускать R, можно иметь любые версии этой операционной системы, начиная с Windows 95. Как и в предыдущем случае, установочный пакет скачивается с CRAN. Установка напоминает обычную установку приложения для Windows. Есть программа-установщик, которая задает несколько простых вопросов, от ответов на которые ничего серьезного не зависит. После инсталляции появляется ярлык, щелкнув на который, можно запускать R.

Здесь интересно заметить, что Windows-инсталляция R является portable и может запускаться, например, с флэшки или лазерного диска. R делает пару записей в реестр, но для работы они совершенно не критичны. Единственное, что надо при этом иметь в виду, рабочая папка должна быть открыта для записи, иначе вам некуда будет записывать результаты работы. Еще одна вещь, важная для всех операционных систем: R (в отличие от S-Plus) держит все свои вычисления в оперативной памяти, поэтому если в процессе работы, скажем, выключится питание, результаты сессии, не записанные явным образом в файл, пропадут. (Правда, существует пакет SOAR, который изменяет это поведение.)

Каждая операционная система имеет свои особенности работы. Но в целом можно сказать, что под все три упомянутые выше операционные системы существует так называемый терминальный способ запуска, а под Mac и Windows имеется свой графический интерфейс (GUI) с некоторыми дополнительными возможностями (разными в разных операционных системах).

Терминальный способ прост: достаточно в командной строке терминала набрать в командной строки, например, Debian:

```
$ R, -
```

и появится вводный экран системы

```
R version 3.1.0 (2014-04-10) -- "Spring Dance"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (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.

> b <- matrix(1:9, ncol=3)
> print(b)
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
>
```

Набираем пример с матрицей.

```
> b <- matrix(1:9, ncol=3)
```

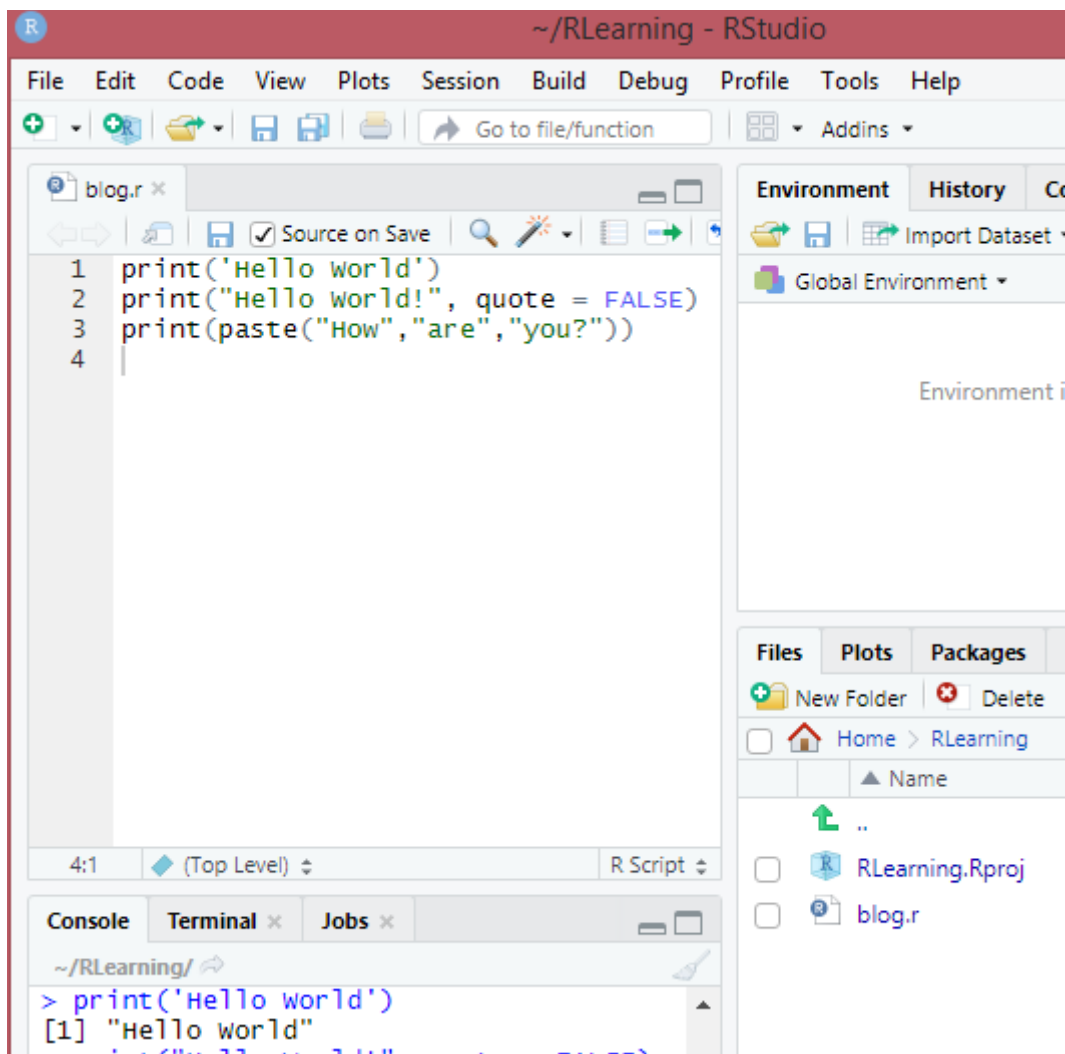
и выводим результат переменной `b` командой `print`.

```
> print(b)
```

Под Windows это немного сложнее необходимо вызывать программу `rgui.exe`, причем для этого надо либо находиться в той же папке, где находится эта программа, либо путь к этой папке должен быть прописан в переменной `PATH`. Кроме того, для того чтобы в русскоязычной Windows вводный экран был читаем, надо сменить в окне терминала кодировку (командой `chcp1251`) и установить соответствующий шрифт (скажем, `Lucida Console`). Если в UNIX терминал запущен без графической среды (X11), то все изображения будут скидываться в один многостраничный PDF-файл `Rplots.pdf`. Под Mac это будет происходить, даже если X11 запущен, так что полноценно использовать R под Mac можно только в GUI-варианте. Терминальный запуск под Windows таких ограничений не имеет.

В дальнейшем договоримся, что под сессией R будет иметься в виду терминальный запуск под X11 в Linux и GUI-запуск в Windows и Mac. GUI под эти операционные системы построены так, что они все равно запускают

терминал-подобное окно - общение с R возможно только в режиме диалога, "команда - ответ" . Полноценного GUI с R не поставляется, хотя существуют многочисленные попытки создать такую систему. Одна из лучших, на наш взгляд, это - R Commander, советуем также обратить внимание на RStudio.



По ссылке <https://rstudio.com/products/rstudio/download/> можно скачать RStudio (справочная информация по установке RStudio <https://rstudio.com/products/rstudio/#rstudio-desktop>).

Но система из меню и окошек не способна заменить полноценного интерфейса командной строки, особенно в случае таких сложных систем, как R. Особенность R GUI под Windows – ее можно запустить в много-(MDI) и однооконном (SDI) режимах. Рекомендуется применять второй, тем более что все остальные реализации R только его и "умеют".

После запуска R в начале появляется вводный экран. Если у Вас русифицированная система, то Вы увидите надпись по-русски. Если нужно

изменить интерфейс на английский язык, установите переменную `LANGUAGE` создайте файл `Renviron.site`, в который внесите строку

```
LANGUAGE=en
```

Этот файл должен находиться в так называемой "домашней" директории R. Более подробно про это можно узнать, если прочитать помощь по команде `Startup()`. Под Linux можно просто внести эту строку в строку вызова как опцию.

Выход из R

Перед тем как начать работать, надо понять, как выйти. Для этого надо ввести одну команду, нажать Enter и ответить на один вопрос:

```
>q()
```

```
Save workspace image? [y/n/c]: n
```

Уже такой простой пример демонстрирует, что в R любая команда имеет аргумент в круглых скобках. Если аргумент не указан, скобки все равно нужны. Если вы забудете это сделать, то вместо выхода из R получите определение функции:

```
> q
```

```
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
```

```
<environment: namespace:base>
```

Как узнать, как правильно вызывать функцию? Для этого надо научиться получать справку. Есть два пути. Первый вызвать команду справки:

```
>help(q)
```

или

```
>?q
```

Вы увидите отдельное окно либо (если вы работаете в Linux) текст помощи в основном окне программы (выход из этого режима по клавише "q"). Если внимательно прочитать текст, становится ясно, что выйти из R можно, и не отвечая на вопрос, если ввести `q("no")`.

Зачем же нужен этот вопрос? Что будет, если ответить положительно? В этом случае в рабочую папку R (ту, из которой он вызван) запишутся два файла: бинарный `.RData` и текстовый `.Rhistory`. Первый содержит все объекты,

созданные вами за время сессии. Второй полную историю введенных команд.

Когда вы работаете в R, предыдущую команду легко вызвать, нажав клавишу-стрелку вверх. Но если вы сохраните файл `.Rhistory`, Ваши команды будут доступны и в следующей сессии - при условии, что Вы вызовете R из той же самой папки. И наоборот, если Вы случайно сохраните рабочую среду (эти два файла), то при следующем старте они загрузятся автоматически. Иногда такое поведение R становится причиной различных недоумений, так что будьте внимательны.

1.1.2. Основные команды

Для выполнения тех или иных операций R использует функции. Для запуска функции с именем `funcname` мы набираем `funcname (input1, input2)`, где входные параметры, или аргументы, `input 1` и `input2` сообщают R, как именно следует исполнить эту функцию. Например, для создания вектора с несколькими числами мы используем функцию конкатенации `c()`. Следующая команда говорит R объединить числа 1, 3, 2 и 5 и сохранить их в виде вектора с именем `x`. Когда мы наберем `x`, то в ответ получим этот вектор.

```
> x <- c(1, 3, 2, 5)
> x
[1] 1 3 2 5
```

Обратите внимание на то, что `>` не является частью команды — R выводит этот знак просто, чтобы показать свою готовность к выполнению следующей команды. Мы можем сохранять объекты не только с помощью `<-`, но также и `=`:

```
> x = c(1, 6, 2)
> x
[1] 1 6 2
> y = c(1, 4, 3)
```

Многократное нажатие клавиши со стрелкой вверх приведет к показу предыдущих команд, которые можно отредактировать. Это полезно, поскольку необходимость повторения похожих команд возникает часто.

Кроме того, ввод команды `?funcname` всегда откроет новое окно со справочным файлом, содержащим дополнительную информацию по функции `funcname`.

Мы можем попросить R выполнить сложение двух чисел. В этом случае программа сначала добавит первое число из x к первому числу из y и т. д. Однако x и y должны быть одинаковой длины. Мы можем проверить их длину при помощи функции `length()`.

```
> length(x)
```

```
[1] 3
```

```
> length(y)
```

```
[1] 3
```

```
> x + y
```

```
[1] 2 10 5
```

Функция `ls()` позволяет просмотреть список всех объектов, таких как данные и функции, которые мы сохранили до этого момента. Функцию `rm()` можно использовать для удаления любого нежелательного объекта.

```
> ls()
```

```
[1] "x" "y"
```

```
> rm(x, y)
```

```
> ls()
```

```
Character (0)
```

Имеется также возможность удалить все объекты за один раз:

```
> rm(list = ls())
```

Функцию `matrix()` можно использовать для создания матрицы с числами. Перед применением функции `matrix()` мы можем узнать о ней больше:

```
> ?matrix
```

Справочный файл сообщает, что функция `matrix()` принимает несколько входных параметров, но пока мы сосредоточимся на первых трех: данные (элементы матрицы), количество строк и количество столбцов. Сначала создадим простую матрицу.

```
> x = matrix (data = c(1, 2, 3, 4), nrow = 2, ncol = 2)
```

```
> x
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

Обратите внимание, что мы могли бы с тем же успехом не набирать `data =`, `nrow =` и `ncol =` в приведенной выше команде `matrix()`, а просто ввести

```
x = matrix(c(1, 2, 3, 4), 2, 2)
```

и это имело бы тот же эффект. Однако иногда бывает полезно указать имена аргументов, т. к. иначе R будет предполагать, что аргументы функции подаются на нее в том же порядке, который приведен в справочном файле этой функции. Как показано в данном примере, по умолчанию R создает матрицы путем последовательного заполнения столбцов. В качестве альтернативы можно использовать опцию `byrow = TRUE` для заполнения матрицы по строкам.

```
> matrix(c(1, 2, 3, 4), 2, 2, byrow = TRUE)
[,1] [,2]
[1,] 1 2
[2,] 3 4
```

Заметьте, что в приведенной выше команде мы не присвоили матрице никакого имени вроде `x`. В этом случае матрица выводится на экран, но не сохраняется для будущих вычислений. Функция `sqrt()` возвращает квадратный корень каждого элемента вектора или матрицы. Команда `x^2` возводит каждый элемент `x` в квадрат; возможно использование любой степени, включая дроби и отрицательные степени.

```
> sqrt(x)
      [,1] [,2]
[1,] 1.00 1.73
[2,] 1.41 2.00
> x^2
      [,1] [,2]
[1,]    1    9
[2,]    4   16
```


Функция `rnorm()` генерирует вектор случайных нормально распределенных значений, при этом аргумент *n* задает размер выборки. Каждый раз при вызове этой функций мы будем получать другой результат. Здесь мы создаем два коррелирующих набора чисел *x* и *y* и применяем `cor()` функцию `cor()` для расчета корреляции между ними.

```
> x = rnorm(50)
> y = x + rnorm(50, mean = 50, sd = .1)
> cor(x, y)
[1] 0.995
```

По умолчанию `rnorm()` создает случайные переменные, представляющие стандартное нормальное распределение со средним значением 0 и стандартным отклонением 1. Однако, как показано выше, среднее значение и стандартное отклонение можно изменить при помощи аргументов `mean` и `sd`. Иногда мы хотим, чтобы наш код в точности воспроизводил один тот же набор случайных чисел; для этого мы можем использовать `set.seed()` функцию `set.seed()`. Функция `set.seed()` принимает в качестве аргумента произвольное целое число.

```
> set.seed(1303)
> rnorm(50)
[1] -1.1440 1.3421 2.1854 0.5364 0.0632 0.5022 -0.0004
```

Мы используем `set.seed()` во всех лабораторных работах каждый раз, когда выполняем вычисления, содержащие случайные величины. Как правило, это должно помочь пользователю воспроизвести наши результаты. Однако следует отметить, что с появлением новых версий R могут возникнуть небольшие несоответствия между книгой и тем, что выдает R.

Функции `mean()` и `var()` можно использовать для вычисления среднего значения и дисперсии некоторого набора чисел. Применение `sqrt()` к результату работы `var()` даст стандартное отклонение, можно просто применить функцию `sd()`.

```
> set.seed(0)
> y = rnorm(100)
> mean(y)
[1] 0.0110
```

```
> var(y)
[1] 0.7329
> sqrt(var(y))
[1] 0.8561
> sd(y)
[1] 0.8561
```

1.2. Написание скриптов на языке R

Задание

Написать скрипт из команд разделов 1.1.1 и 1.1.2, сохранить и запустить скрипт. Подключить пакет и запустить функцию из пакета.

Просто открыть сессию R и вводить в окно программы команды, одну за другой — это лишь один из возможных способов работы. Гораздо более продуктивный метод, который является заодно и серьёзнейшим преимуществом R — это создание скриптов. Иначе говоря, программ, которые потом загружаются в R и интерпретируются им. С самого начала работы следует создавать скрипты, даже для таких задач, которые кажутся пустяковыми — это в будущем значительно сэкономит Ваше время.

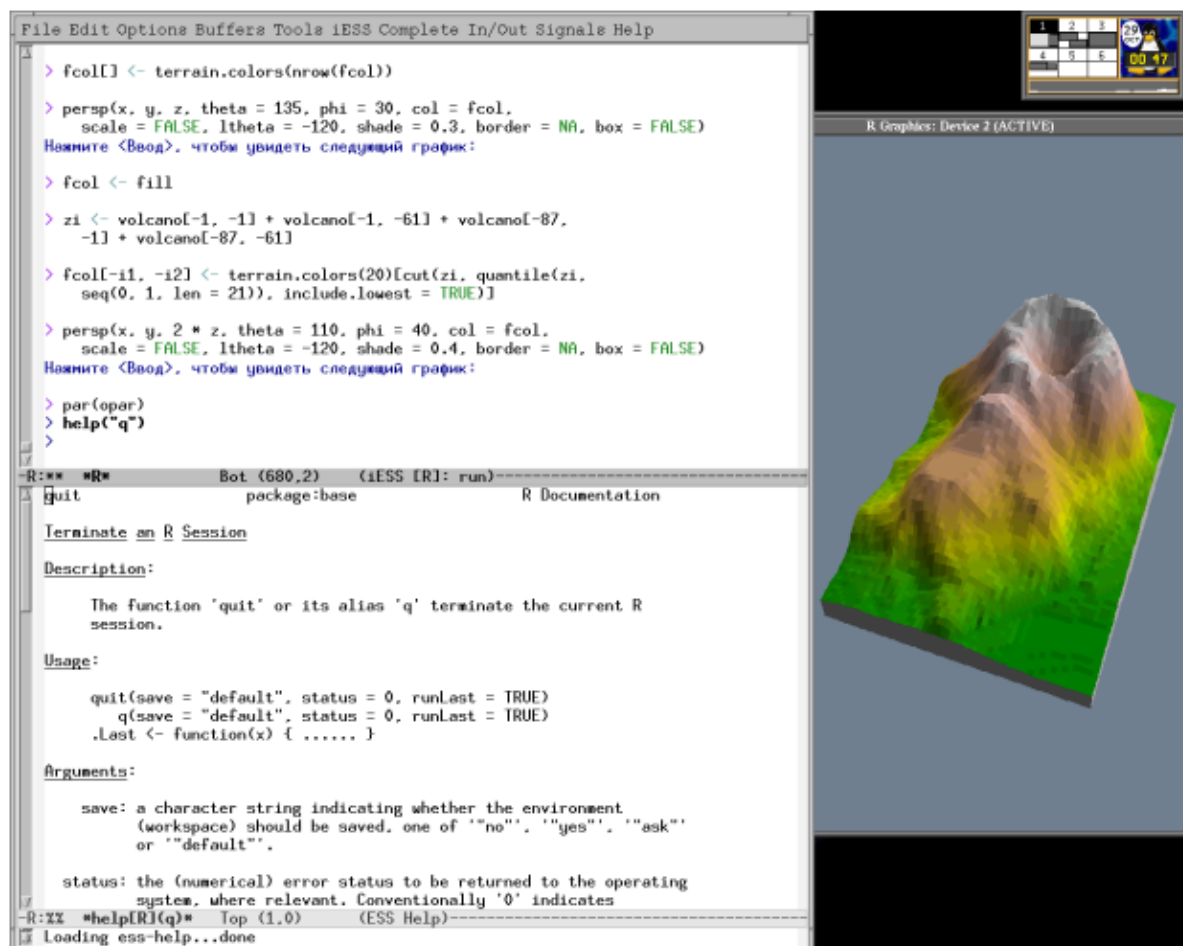
Создание скриптов по любому поводу и даже без особого повода — одна из основ культуры работы в R. Создать скрипт очень просто. Допустим, после открытия сессии была введена необходимая для получения искомого результата последовательность команд. Чтобы сделать свою работу воспроизводимой — надо просто сохранить историю команд. Лучше всего это сделать с помощью команды:

```
> savehistory(file="myscript.r")
```

После этого в текущей папке появляется файл `myscript.r`, который можно отредактировать любимом текстовом редакторе, а потом загрузить в R командой:

```
> source("myscript.r", echo=TRUE)
```

Опция `echo` добавлена для того чтобы можно было видеть сами команды, а не только результат их выполнения. Есть ещё более эффективный способ работы: Вы открываете Ваш скрипт в текстовом редакторе, а потом посылаете отдельные его строки прямо в R. Есть несколько редакторов, которые умеют так делать. Во-первых, это Emacs с установленным пакетом ESS («Emacs Speaks Statistics»).



Преимущество этой системы в том, что R запускается прямо в одном из окон редактора. Во-вторых, штатные R GUI под Windows и под Mac также имеют встроенные редакторы скриптов. К сожалению, Windows-редактор не подсвечивает синтаксис, и вообще довольно неудобен. Вместо него тем пользователям, которых пугает перспектива освоения Emacs, можно порекомендовать отличный редактор Tinn-R, специально «заточенный» под такую работу.

Скрипт R можно выполнить и не запуская интерактивную сессию. Для этого используются специальные опции командной строки. Вот так можно, например, выполнить наш скрипт-пример:

```
=> R --no-save < myscript.r > out
```

Опция `--no-save` говорит R не сохранять результаты сессии в файле истории `.RData/.Rhistory` (фактически, отвечает «no» на упомянутый выше заключительный вопрос).

Пакеты. Ещё одно важное преимущество R — наличие для него многочисленных расширений или пакетов буквально на все случаи жизни. При установке R на компьютер, несколько пакетов уже в наличии: так

называемые базовые пакеты, без которых система просто не работает (пакет базовых функций `base` или пакет `grDevices`, который управляет выводом графиков), и «рекомендованные» пакеты (пакет для специализированного кластерного анализа `cluster`, пакет для анализа нелинейных моделей `nlme` и пр.). Кроме того можно поставить любой из почти полутора тысяч (!) доступных на CRAN пакетов.

При доступном Интернете, это можно сделать прямо из R командой `install.packages()` (а под Mac и Windows есть соответствующие пункты в меню). Если соединение с сетью низкоскоростное, то можно скачать исходные тексты пакетов (под GNU/Linux) или скомпилированные пакеты (под Mac или Windows) и установить прямо с диска. После скачивания исходников пакета перед использованием сначала его нужно скомпилировать, потому что многие пакеты содержат код на FORTRAN или C. Для этого есть специальная форма вызова R:

```
=> R CMD INSTALL package.tar.gz
```

Естественно, это всё следует делать, если R устанавливается не и из стандартного репозитория дистрибутива GNU/Linux. В случае стандартной установки можно поискать пакеты по сочетанию `cran`. В Debian GNU/Linux Etch таких пакетов ровно 85 — это не 1500 пакетов с CRAN, но скорее всего там уже есть многое из того, что необходимо. Как только пакет установлен, то он сразу готов к работе. Нужно только инициализировать его перед употреблением. Для этого служит команда `library()`.