Paketbau in R

Hans Peter Wolf

August 2008, gedruckt: November 12, 2008

Inhaltsverzeichnis

1 Einleitung

1.1 Pakete – warum eigentlich?

Pakete dienen dazu, fertige Funktionen und Datenobjekte zusammengepackt abzulegen. Objekte können zwar auch als Ascii-Dateien oder binär gespeichert werden, doch weist die Organisationsform Paket Vorteile auf. Ein wesentlicher besteht darin, dass Objekte in Paketen nur zusammen mit Help-Seiten zusammengefasst werden können. Weiterhin lassen sich Pakete für bestimmte Gruppen von Funktionen definieren, wobei sich bei gegenseitigen Abhängigkeiten von Funktionen innerhalb eines Paketes bei Nutzung des Namespace-Konzeptes keine Konflikte zu anderen Paketen ergeben. Als letzter Vorteil sei hier aufgeführt, dass Pakete die zentrale Form darstellen, um R-Lösungen anderen R-Nutzern zur Verfügung zu stellen. Denn durch einfache Upload-, Download- sowie lokale Ladevorgänge lassen sich Pakete propagieren und nutzen. So können R-User inzwischen weit über 1500 Pakete über Netz beschaffen, in ihr R einbauen und deren Funktionalitäten mittels `library` öffnen.

1.2 Pakete – wie konstruiert und nutzt man sie?

Zur Konstruktion eines eigenen Paketes sind folgende Schritte zu erledigen:

1. Definition von Funktionen und Datenobjekten
2. Erstellung von Hilfeseiten für Objekte
3. Konstruktion eines Paketverzeichnisses
4. Überarbeitung des DESCRIPTION-File und der Paket-Help-Seite
5. Platzierung der neuen Dateien aus 1. bis 2. ins Paketverzeichnis
6. Check und Bau des Paketes
7. Installation des Paketes (durch Anwender)
8. Nutzung des Paketes (durch Anwender)

Als ergänzende Diskussionspunkte sollen erwähnt werden

- Paketbau unter Windows
- Einsatz von Namespaces
- Integration von C-Programmen
- Upload auf CRAN

Zur Demonstration des Paketbaus verpacken wir im nächsten Kapitel eine Funktion mit dem Namen `word.count`, die die Worthäufigkeiten der Wörter einer Datei ermittelt. Dabei sind verschiedene Klippen zu überwinden, deren Umschiffung kommentiert wird. Das Paket soll den Namen `wc` bekommen.

Für den schnellen Überblick skizzieren wir aber zuvor den Weg für die kleine Funktion `word.count1`, die in einem Paket `wc1` untergebracht werden soll. So bekommt der Neuling eine kleine Vorstellung und Erfahrene können sich schnell wieder an den Gang der Dinge erinnern.

1.3 Minimalbeispiel zur Paketerstellung

Gemäß den oben vorgestellten Punkten durchlaufen wir in diesem Abschnitt schnell die Tour zur Paketerstellung.

1. Zunächst definieren wir die Funktion `word.count1`. Dieser Schritt ist in der Regel schon erledigt, wenn man Pakete definiert.

1 ⟨* 1⟩ ≡
```
word.count1<<-function(txt){ table(txt) }
```

2. Ein Rohentwurf einer Hilfeseite zu einer Funktion wird von der R-Funktion `prompt` erstellt und in einer `Rd`-Datei (hier `word.count1.Rd`) abgelegt. Die Inhalte dieser Datei müssen dann mit einem Editor korrigiert werden.

2 ⟨* 1⟩+ ≡
```
prompt(word.count1)
```

3. Die Verzeichnisstruktur für ein Paket wird durch die R-Funktion `package.skeleton` aufgebaut. Default-mäßig werden die Objekte in `.GobalEnv` erwartet.

3 ⟨* 1⟩+ ≡
```
package.skeleton(name = "wc1", list="word.count1", path = ".")
```

4. Der von `package.skeleton` angelegte `DESCRIPTION`-File muss per Texteditor überarbeitet werden. Gleichfalls muss der Paket-Help-File `wc1-package.Rd` im Verzeichnis `man` angepasst werden.

5. Die Dateien mit Objektdefinitionen und die Helpseiten müssen im Paketverzeichnis untergebracht werden. `package.skeleton` platziert die gefundenen Definition für Objekte in einem `R`-File im Unterverzeichnis `R`. Bei Veränderung müssen die entsprechenden Definitionen angepasst werden. Wir aktualisieren die Hilfeseite:

4 ⟨* 1⟩+ ≡
```
file.copy(from="word.count1.Rd",
          to="wc1/man/word.count1.Rd"),overwrite=TRUE)
```

6. Check und Bau des Paketes:

5 ⟨* 1⟩+ ≡
```
system("R CMD check wc1")
system("R CMD build wc1")
```

7. Für die Installation des Pakets in einem existierenden Unterverzeichnis, in der Bibliothek `mylib`, genügt folgender Aufruf:

6 ⟨* 1⟩+ ≡
```
system(paste("R CMD INSTALL --library=mylib wc1_0.1.tar.gz",sep=""))
```

8. Die Inhalte des Pakets lassen sich nach Installation vom Anwender nutzen.

7 ⟨* 1⟩+ ≡
```
library(wc1,lib.loc="mylib")
```

Leider läuft in der Regel nicht alles glatt. Deshalb wird es für viele hilfreich sein, die ausführliche Wegbeschreibung zu studieren. Ergänzend sei bemerkt, dass unter Windows zuvor noch einige Installationshandgriffe zu erledigen sind.

1.4 Was folgt

Im nächsten Abschnitt üben wir den Umgang mit Bibliotheken und Paketen. Dann verschnüren wir die Demo-Funktion `word.count` in dem Paket `wc` und installieren es an einer speziellen Stelle. Zunächst zeigen wir die Vorgehensweise auf einer Linux-Maschine. Die Unterschiede zu Windows werden in einem gesonderten Abschnitt beschrieben.

1.5 Bibliotheken und Pakete – wie gehen wir mit diesen um?

Zuerst frischen wir unser Wissen im Umgang mit Paketen auf. Fertige Funktionen werden in Paketen (packages) zusammengefasst. (Mehrere) Pakete werden in einer Bibliothek (library) abgelegt. Zum Beispiel enthält das Basispaket elementare R-Funktionen wie `ls()`, graphische Routinen sind in dem Paket `graphics` und statistische in `stats` zu finden. Diese Pakete befinden sich in der Bibliothek `library` der R-Software. Welche Bibliotheken erreichbar sind, ist der Funktion `.libPaths()` zu entnehmen, mit der auch eine weitere Bibliothek dem Suchpfad der Bibliothek hinzugefügt werden kann.

8 ⟨* 1⟩+ ≡
```
.libPaths()
```

Wir erhalten die Antwort:

```
[1] "/usr/lib/R/library"
```

Welche Pakete in einer Bibliothek vorhanden sind, zeigt uns `library()` an.

9 ⟨* 1⟩+ ≡
```
library()
```

Das Ergebnis könnte so aussehen:

```
Packages in library '/usr/lib/R/library':

KernSmooth              Functions for kernel smoothing for Wand &
                        Jones (1995)
MASS                    Main Package of Venables and Ripley's MASS
base                    The R Base Package
boot                    Bootstrap R (S-Plus) Functions (Canty)
class                   Functions for Classification
cluster                 Functions for clustering (by Rousseeuw et al.)
ctest                   Defunct Package for Classical Tests
eda                     Defunct Package for Exploratory Data Analysis
foreign                 Read data stored by Minitab, S, SAS, SPSS,
                        Stata, ...
graphics                The R Graphics Package
...
```

Falls die fragliche Bibliothek an anderer Stelle zu finden ist, kann das über das Argument `lib.loc` der Funktion `library()` vermittelt werden:

10 ⟨* 1⟩+ ≡
```
library(lib.loc="mylib")
```

Mit `library()` lassen sich auch Pakete laden. So lädt folgender Befehl das Paket `boot`.

11 ⟨* 1⟩+ ≡
```
library(boot)
```

Als Antwort wird die Liste der geladenen Pakete ausgegeben. Durch das Laden eines Paketes wird der Suchpfad, anhand dessen ggf. nach Objekten gesucht wird, verlängert.

12 ⟨* 1⟩+ ≡
```
search()
```

liefert:

```
 [1] ".GlobalEnv"        "package:boot"      "package:relax"
 [4] "package:tcltk"     "package:stats"     "package:graphics"
 [7] "package:grDevices" "package:utils"     "package:datasets"
[10] "package:methods"   "Autoloads"         "package:base"
```

Die zugehörigen Pfade, auf denen nach Objekten gesucht wird, liefert:

13 ⟨* 1⟩+ ≡
```
searchpaths()
```

Beispielsweise könnte dieses zu folgendem Output führen

```
 [1] ".GlobalEnv"                   "/usr/lib/R/library/boot"
 [3] "/usr/lib/R/library/relax"     "/usr/lib/R/library/tcltk"
 [5] "/usr/lib/R/library/stats"     "/usr/lib/R/library/graphics"
 [7] "/usr/lib/R/library/grDevices" "/usr/lib/R/library/utils"
 [9] "/usr/lib/R/library/datasets"  "/usr/lib/R/library/methods"
[11] "Autoloads"                    "/usr/lib/R/library/base"
```

Die Pfade zu den Paketen bilden eine Teilmenge der ersten Ausgabe, und wir erhalten diese durch:

14 ⟨* 1⟩+ ≡
```
.path.package()
```

```
 [1] "/usr/lib/R/library/boot"      "/usr/lib/R/library/relax"
 [3] "/usr/lib/R/library/tcltk"     "/usr/lib/R/library/stats"
 [5] "/usr/lib/R/library/graphics"  "/usr/lib/R/library/grDevices"
 [7] "/usr/lib/R/library/utils"     "/usr/lib/R/library/datasets"
 [9] "/usr/lib/R/library/methods"   "/usr/lib/R/library/base"
```

Ein geladenes Paket wird wieder entfernt über den Namen oder die Position im Suchpfad durch die Anweisung:

15 ⟨* 1⟩+ ≡
```
detach("package:boot")
search()
```

```
 [1] ".GlobalEnv"        "package:relax"     "package:tcltk"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
```

Von der Position im Suchpfad zur Umgebung führt uns die Funktion `pos.to.env`:

16 ⟨* 1⟩+ ≡
```
library(boot)
print(pos.to.env(2))
```

Ergebnis:

```
<environment: package:boot>
attr(,"name")
[1] "package:boot"
attr(,"path")
[1] "/usr/lib/R/library/boot"
```

2 Der Paketbau anhand eines kleinen Beispiels

2.1 Objekte – wie erstellen wir Objekte für ein Paket?

Wir erstellen Objekte für ein Paket, wie wir üblicherweise Objekte erstellen. Zur Demonstration schreiben wir eine Funktion mit dem Namen `word.count`, die die Häufigkeiten von Wörtern einer Datei bestimmt. Diese wollen wir in einem Paket mit dem Namen `wc` ablegen.

17 ⟨* 1⟩+ ≡
```
word.count<-
function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-gsub(" +"," ",txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
}
```

Ein Test der Funktion sei gleich angeschlossen – probieren Sie selbst!

18 ⟨* 1⟩+ ≡
```
word.count(fname="create-packages.rev")
```

Sie müssen natürlich hinter `fname` eine existierende Datei benennen.

2.2 Help-Page – wie erstellen wir zu einer Funktion eine Hilfeseite?

Nach der Definition einer Funktion muss an die Hilfe-Seite gedacht werden. Also erstellen wir mit der Funktion `prompt` einen Rohling, der in einer `Rd`-Datei abgelegt wird.

19 ⟨* 1⟩+ ≡
```
## prompt(word.count)
## auskommentiert, um nicht fertigen Rd-file zu ueberschreiben
```

Es entsteht bei Umsetzung des Befehls die Datei `word.count.Rd`:

```
\name{word.count}
\alias{word.count}
%- Also NEED an '\alias' for EACH other topic documented here.
\title{ ~~function to do ... ~~ }
\description{
  ~~ A concise (1-5 lines) description of what the function does. ~~
}
\usage{
word.count(fname)
}
%- maybe also 'usage' for other objects documented here.
\arguments{
  \item{fname}{ ~~Describe \code{fname} here~~ }
}
\details{
  ~~ If necessary, more details than the description above ~~
}
\value{
  ~Describe the value returned
  If it is a LIST, use
  \item{comp1 }{Description of 'comp1'}
  \item{comp2 }{Description of 'comp2'}
  ...
}
\references{ ~put references to the literature/web site here ~ }
\author{ ~~who you are~~ }
\note{ ~~further notes~~
```

```
 ~Make other sections like Warning with \section{Warning }{....} ~
}
\seealso{ ~~objects to See Also as \code{\link{help}}, ~~~ }
\examples{
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-gsub(" +"," ",txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
  }
}
% Add one or more standard keywords, see file 'KEYWORDS' in the
% R documentation directory.
\keyword{ ~kwd1 }
\keyword{ ~kwd2 }% __ONLY ONE__ keyword per line
```

Wir modifizieren per Editor den Inhalt der Roh-Hilfeseite. Dabei verwenden wir die deutsche Sprache, damit man schnell selbst Hinzugefügtes erkennen kann. Für eine breite Weitergabe fertiger Pakete sind Texte in englischer Sprache vorzuziehen. Probleme könnte die Identifikation eines geeigneten Keyword sein (siehe am Ende der Hilfeseite). Verwendete Keywords findet man übrigens in der Datei `KEYWORDS`:

20 ⟨ * 1⟩+ ≡

```
 file.show( file.path(R.home(),"doc","KEYWORDS") )
```

```
GROUPED Keywords
----------------

Graphics
        aplot           &       Add to Existing Plot / internal plot
        dplot           &       Computations Related to Plotting
        hplot           &       High-Level Plots
        iplot           &       Interacting with Plots
        color           &       Color, Palettes etc
        dynamic         &       Dynamic Graphics
        device          &       Graphical Devices

Basics
        sysdata         &       Basic System Variables          [!= S]
        datasets        &       Datasets available by data(.)   [!= S]
        data            &       Environments, Scoping, Packages [~= S]
        manip           &       Data Manipulation
        attribute       &       Data Attributes
        classes         &       Data Types (not OO)
            & character &       Character Data ("String") Operations
            & complex   &       Complex Numbers
            & category  &       Categorical Data
            & NA        &       Missing Values                  [!= S]
        list            &       Lists
        chron           &       Dates and Times
        package         &       Package Summaries

Mathematics
        array           &       Matrices and Arrays
              & algebra &       Linear Algebra
```

```
        arith           &       Basic Arithmetic and Sorting    [!= S]
        math            &       Mathematical Calculus etc.      [!= S]
        logic           &       Logical Operators
        optimize        &       Optimization
        symbolmath      &       "Symbolic Math", as polynomials, fractions
        graphs          &       Graphs, (not graphics), e.g. dendrograms

Programming, Input/Ouput, and Miscellaneous

        programming     &       Programming
             & interface&       Interfaces to Other Languages
        IO              &       Input/output
             & file     &       Files
             & connection&      Connections
             & database &       Interfaces to databases
        iteration       &       Looping and Iteration
        methods         &       Methods and Generic Functions
        print           &       Printing
        error           &       Error Handling

        environment     &       Session Environment
        internal        &       Internal Objects (not part of API)
        utilities       &       Utilities
        misc            &       Miscellaneous
        documentation   &       Documentation
        debugging       &       Debugging Tools

Statistics

        datagen         &       Functions for generating data sets
        distribution    &       Probability Distributions and Random Numbers
        univar          &       simple univariate statistics  [!= S]
        htest           &       Statistical Inference
        models          &       Statistical Models
            & regression&       Regression
            & &nonlinear&       Non-linear Regression (only?)
        robust          &       Robust/Resistant Techniques
        design          &       Designed Experiments
        multivariate    &       Multivariate Techniques
        ts              &       Time Series
        survival        &       Survival Analysis
        nonparametric   &       Nonparametric Statistics [w/o 'smooth']
        smooth          &       Curve (and Surface) Smoothing
             & loess    &       Loess Objects
        cluster         &       Clustering
        tree            &       Regression and Classification Trees
        survey          &       Complex survey samples
```

Hier ist die modifizierte Version der Help-Seite von `word.count`:

```
\name{word.count}
\alias{word.count}
%- Also NEED an '\alias' for EACH other topic documented here.
\title{ W\"orterz\"ahlfunktion }
\description{
  \code{word.count} z\"ahlt die H\"aufigkeiten von W\"ortern einer Datei.
}
\usage{
word.count(fname)
}
%- maybe also 'usage' for other objects documented here.
\arguments{
  \item{fname}{ Name der zu untersuchenden Datei }
}
\details{
  \code{word.count} liest den Inhalt der Datei, ermittelt die W\"orter und
  z\"ahlt die H\"aufigkeiten.
}
\value{
  Ausgegeben wird ein Vektor der H\"aufigkeiten. Die W\"orter werden
    als Namen der Objekte verwendet.
```

```
}
\references{ vgl. Benford's Law:: \link{http://en.wikipedia.org/wiki/Benford%27s_law} }
\author{ P. Wolf }
\note{
  \section{Warning }{Es lassen sich nur Text-Dateien untersuchen}
}
\seealso{ \code{\link{table}}, \code{\link{scan}} }
\examples{
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
word.count("testdatei")

## The function is currently defined as
function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-gsub(" +"," ",txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
  }
}
% Add one or more standard keywords, see file 'KEYWORDS' in the
% R documentation directory.
\keyword{ IO }
```

2.3 Paketverzeichnis – wie entwerfen wir die Grundstruktur?

2.3.1 `package.skeleton` für das Grobgerüst

Mit der Funktion `package.skeleton` lässt sich bequem eine Verzeichnisstruktur aufbauen, die für die Paket-Konstruktion notwendig ist. Hier ist ein Ausschnitt der Hilfeseite zu `package.skeleton`:

```
Description:

     'package.skeleton' automates some of the setup for a new source
     package.  It creates directories, saves functions, data, and R
     code files to appropriate places, and creates skeleton help files
     and a 'Read-and-delete-me' file describing further steps in
     packaging.

Usage:

     package.skeleton(name = "anRpackage", list,
                      environment = .GlobalEnv,
                      path = ".", force = FALSE, namespace = FALSE,
                      code_files = character())

Arguments:

    name: character string: the package name and directory name for
          your package.

    list: character vector naming the R objects to put in the package.
          Usually, at most one of 'list', 'environment', or
          'code_files' will be supplied.  See details.

environment: an environment where objects are looked for. See details.

    path: path to put the package directory in.

   force: If 'FALSE' will not overwrite an existing directory.
```

```
namespace: a logical indicating whether to add a name space for the
          package.  If 'TRUE', a 'NAMESPACE' file is created to export
          all objects whose names begin with a letter, plus all S4
          methods and classes.

code_files: a character vector with the paths to R code files to build
          the package around.  See details.
...
```

Wir wollen die Baustelle in dem Verzeichnis `test` unter dem aktuellen Verzeichnis einrichten. Da wir verschiedentlich mit Pfaden hantieren müssen, legen wir diese auf Variablen ab. Wir legen folgende Verzeichnisse fest:

21 ⟨*start* 21⟩ ≡
```
wdpath<-getwd()
testpath<-file.path(wdpath,"test")
libpath<-file.path(wdpath,"test","mylib")
```

Wir legen unter unserer Arbeitsstelle `wdpath` ein Testverzeichnis `test` an. Den Pfad zu diesem Verzeichnis hatten wir gerade auf `testpath` abgelegt. In diesem soll das neue Paket mit dem Namen `wc` entstehen.

22 ⟨* 1⟩+ ≡
```
if(!file.exists(testpath)) dir.create(testpath)
```

Nun starten wir die Gerüst-Erzeugungs-Funktion, verwenden jedoch nicht alle Parameter. Weitere Erklärungen entnehme man der Help-Seite von `package.skeleton`.

23 ⟨* 1⟩+ ≡
```
setwd(wdpath)
package.skeleton(name = "wc", list="word.count",
                 path = testpath)
"Paketstruktur angelegt"
```

Als Ergebnis liegt nun in dem Verzeichnis `test` ein Unterverzeichnis `wc` vor, das folgende Dinge enthält:

24 ⟨* 1⟩+ ≡
```
cat(list.files(file.path(testpath,"wc"),recursive=TRUE),sep="\n")
```

Wir erhalten folgende Auflistung:

```
DESCRIPTION
man/wc-package.Rd
man/word.count.Rd
Read-and-delete-me
R/word.count.R
```

Alternativ hätten wir die Verzeichnisstruktur auch per Hand bilden können durch ...

25 ⟨* 1⟩+ ≡
```
dir.create(file.path(testpath,"wc"))
dir.create(file.path(testpath,"wc/man"))
dir.create(file.path(testpath,"wc/R"))
```

... und hätten dann die Dateien `DESCRIPTION`, `man/wc-package.Rd`, `man/word.count.Rd` und `R/word.count.R` ebenfalls manuell platzieren müssen.

2.3.2 `Read-and-delete-me` für die nächsten Schritte

In `Read-and-delete-me` werden wohl wichtige Infos stehen, so dass wir diese einmal einblenden sollten:

26 ⟨* 1⟩+ ≡
```
cat(scan(file.path(testpath,"wc/Read-and-delete-me"),"",sep="\n"),sep="\n")
```

Wir erfahren:

```
* 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.
Read "Writing R Extensions" for more information.
```

Aha, wir sollen also zuerst die Help-Files erstellen.

2.4 Paketverzeichnis – wie füllen wir die Paket-Grundstruktur?

2.4.1 Die Ablage von Objekt-Help-Dateien

Zur Zeit finden wir – wie schon oben gesehen – ...

27 ⟨* 1⟩+ ≡
```
list.files(file.path(testpath,"wc/man"))
```

... im `man`-Verzeichnis die Dateien:

```
[1] "wc-package.Rd" "word.count.Rd"
```

Für unsere Funktion `word.count` wurde (erneut) ein Help-Rohling erzeugt. Diesen können wir durch unsere bereits verbesserte Version ersetzen:

28 ⟨* 1⟩+ ≡
```
file.copy(from="word.count.Rd",
          to=file.path(testpath,"wc/man/word.count.Rd"),overwrite=TRUE)
"eigene word.count-Help-Seite abgelegt"
```

Die zweite Datei `wc-package.Rd` sollte mit einer Beschreibung des Pakets gefüllt werden. Diese Aufgabe sei dem Leser anvertraut. Im Folgenden verwenden wir zur Demonstration zunächst ungeänderten Datei.

2.4.2 `DESCRIPTION`: Die Beschreibung zum Pakets

Da wir keinen C-Code und auch keine Compilate verwenden, sind für uns der zweite und dritte Hinweis aus dem Readme-File nicht relevant.

Jedoch sollten wir auf jeden Fall den File `DESCRIPTION` modifizieren. Vielleicht in folgender Form. (Hinweis: In der folgenden Datei stehen Umlaute und nicht `\"a` o.ä.)

29 ⟨* 1⟩+ ≡
```
DESCR<-c(
 "Package: wc",
 "Type: Package",
 "Title: Wortz\"ahlungen",
 "Version: 0.1",
 "Date: 2008-08-20",
 "Author: P. Wolf",
 "Maintainer: P. Wolf <pwolf@uni.bielefeld.de>",
 "Description: Funktion zur Ermittlung von H\"aufigkeiten von W\"ortern",
 "License: GPL version 2 or newer",
 "URL: http://www.wiwi.uni-bielefelde.de/~wolf",
 "LazyLoad: yes")
cat(DESCR,sep="\n",file=file.path(testpath,"wc","DESCRIPTION"))
"DESCRIPTION-File erzeugt"
```

2.5 Der erste Check

Bevor wir uns in das gefährliche Bau-Abenteuer stürzen, werden wir zunächst einmal einen ersten Check wagen.

30 ⟨* 1⟩+ ≡

```
setwd(testpath)
system("R CMD check wc")
setwd(wdpath)
# entspricht in "test" aus der Linux Konsole:
# R CMD check wc
```

Wir beobachten folgende Ausgaben.

```
* checking for working pdflatex ... OK
* using log directory '/home/wiwi/pwolf/R/work/test/wc.Rcheck'
* using R version 2.7.0 (2008-04-22)
* using session charset: ISO8859-1
* checking for file 'wc/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'wc' version '0.1'
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking whether package 'wc' can be installed ... OK
* checking package directory ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking DESCRIPTION meta-information ... WARNING
Unknown encoding with non-ASCII data
Fields with non-ASCII values:
  Title Description
See the information on DESCRIPTION files in section 'Creating R
packages' of the 'Writing R Extensions' manual.

* checking top-level files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking for unstated dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... WARNING
Rd files with likely Rd problems:
Unaccounted top-level text in file 'wc-package.Rd':
Following section 'references':
"
  ~~ Optionally other standard keywords, one per line, from file KEYWORDS in ~~
  ~~ the R documentation directory ~~
"
Rd files with unknown encoding:
  word.count.Rd

See the chapter 'Writing R documentation files' in manual 'Writing R
Extensions'.
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... OK
* creating wc-Ex.R ... OK
* checking examples ... ERROR
Running examples in 'wc-Ex.R' failed.
The error most likely occurred in:

> ### * wc-package
```

```
>
> flush(stderr()); flush(stdout())
>
> ### Name: wc-package
> ### Title: What the package does (short line) ~~ package title ~~
> ### Aliases: wc-package wc
> ### Keywords: package
>
> ### ** Examples
>
> ~~ simple examples of the most important functions ~~
Fehler: Unerwartetes Symbol in "~~ simple examples"
Ausf\"uhrung angehalten
```

Oh, oh, oh. Wir erkennen: Der Check war nicht erfolgreich. Auch offenbart eine Inspektion des Dateibaums, dass der Prozess ein neues Verzeichnis angelegt hat. Unter `test` ist jetzt `wc.Rcheck` zu finden. In diesem wird unter dem Unterverzeichnis `wc` probeweise das Paket aufgebaut. Für uns ist in `wc.Rcheck` besonders die Datei `00check.log` interessant, in der das Protokoll des Check-Prozesses zu finden ist. Es ist bei unerklärlichen Fehlern empfehlenswert das Verzeichnis `wc.Rcheck` zu löschen, damit man bei dem nächsten Versuch eine saubere Ausgangssituation hat und keine seltsamen Folgefehler auftreten können. (Man denke zum Beispiel bei LaTeX an eine fehlerhafte `aux`-Datei, die selbst nach Behebung der Ursache weiter Verwirrung stiften kann.)

Wir erkennen, dass der Check stufenweise verläuft. Auf jeder Stufe können Probleme auftreten. Die häufigsten dürften bei der Erstellung der Hilfeseiten auftreten. Denn zum Beispiel muss LaTeX sauber durchlaufen und LaTeXniker wissen, dass selbst kleine Syntax-Verstöße zu Fehlermeldungen führen. Aber auch bei der Programmierung gibt es eine Fülle von Möglichkeiten, Fehler oder Warnungen hervorzurufen. Zum Beispiel wird die Verwendung von `T` für `TRUE` angemerkt. Zur Illustration haben wir schon ein paar Problemsituationen (s.o.) vorliegen, an denen wir den Reparatur-Prozess bis zur Lösungen verfolgen können.

2.6 Fehlersuche und -behebung

Grundsätzlich gilt: Suche die erste Problemstelle und studiere die knappe Meldung intensiv. Andernfalls wird man wahrscheinlich viel zu viele Fehlversuche benötigen. Wir schauen uns demgemäß die Warnungen und Fehlermeldungen nacheinander an.

2.6.1 Nicht-Ascii-Zeichen im Paket-Help-File

Die erste Warnung deutet auf Zeichenprobleme im `DESCRIPTION`-File hin:

```
* checking DESCRIPTION meta-information ... WARNING
Unknown encoding with non-ASCII data
Fields with non-ASCII values:
  Title Description
See the information on DESCRIPTION files in section 'Creating R
packages' of the 'Writing R Extensions' manual.
```

Das können an den verwendeten Umlauten liegen und wir modifizieren:

31 ⟨ * 1⟩+ ≡

```
 DESCR<-c(
  "Package: wc",
  "Type: Package",
  "Title: Wortzaehlungen",
  "Version: 0.1",
  "Date: 2008-08-20",
  "Author: P. Wolf",
  "Maintainer: P. Wolf <pwolf@uni.bielefeld.de>",
  "Description: Funktion zur Ermittlung von Haeufigkeiten von Woertern",
```

```
  "License: GPL version 2 or newer",
  "URL: http://www.wiwi.uni-bielefelde.de/~wolf",
  "LazyLoad: yes")
 cat(DESCR,sep="\n",file=file.path(testpath,"wc","DESCRIPTION"))
 "DESCRITION-File: Umlaute entfernt"
```

Erneutes Checken:

32 ⟨ * 1⟩+ ≡

```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

So, nun läuft bei einem erneuten Check alles glatt bis zur folgenden Stelle.

```
...
* checking Rd files ... WARNING
Rd files with likely Rd problems:
Unaccounted top-level text in file 'wc-package.Rd':
Following section 'references':
"\n~~ Optionally other standard keywords, one per line, from file KEYWORDS in ~~\n~~ the R documentatior

Rd files with unknown encoding:
  word.count.Rd

See the chapter 'Writing R documentation files' in manual 'Writing R
Extensions'.
```

2.6.2 Nicht entfernter Hinweiskommentar im Paket-Help-File

Die Warnung bezieht sich auf den `wc-package.Rd`-File. Offensichtlich müssen aus unserer Paket-Beschreibung zumindest die Zeilen

```
~~ Optionally other standard keywords, one per line, from file KEYWORDS in ~~
~~ the R documentation directory ~~
```

entfernt werden. Per Hand ganz einfach, hier per Programm nur zur Demo.

33 ⟨ * 1⟩+ ≡

```
txt<-scan(file=file.path(testpath,"wc","man","wc-package.Rd"),"",sep="\n")
ind<-charmatch("~~ Optionally other standard keywords",txt)
if(0<length(ind)) txt<-txt[-ind]
ind<-charmatch("~~ the R documentation directory ~~",txt)
if(0<length(ind)) txt<-txt[-ind]
cat(file=file.path(testpath,"wc","man","wc-package.Rd"),txt,sep="\n")
"Paket-Help-Seite: Keyword-Zeilen entfernt"
```

Checken zeigt, dass die Veränderung gewirkt hat:

34 ⟨ * 1⟩+ ≡

```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

2.6.3 Nicht-Ascii-Zeichen im Objekt-Help-File

Im File `word.count.Rd` gibt es ebenfalls Buchstabenprobleme, so dass wir diese durch Umlautvermeidung beheben können. Hätten wir gleich eine englische Dokumentation angestrebt, wäre uns dieses Problem erst gar nicht begegnet.

```
...
* checking Rd files ... WARNING
...
Rd files with unknown encoding:

See the chapter 'Writing R documentation files' in manual 'Writing R
Extensions'.
```

Wieder entfernen wir die Umlaute.

35 ⟨* 1⟩+ ≡
```
txt<-scan(file=file.path(testpath,"wc","man","word.count.Rd"),"",sep="\n")
txt<-gsub("\"a","ae",txt)
txt<-gsub("\"o","oe",txt)
cat(file=file.path(testpath,"wc","man","word.count.Rd"),txt,sep="\n")
"work.count-Help-Seite: Umlaute entfernt"
```

Wir machen einen neuen Check:

36 ⟨* 1⟩+ ≡
```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

So, auch dieser Fehler ist behoben.

2.6.4 Nicht ausführbares Beispiel im Objekt-Help-File

Fehler in der Beispielanwendung. Stimmt, die im Beispielaufruf eingesetzte Datei ist nicht da.

```
...
* checking examples ... ERROR
Running examples in 'wc-Ex.R' failed.
The error most likely occurred in:

> ### * word.count
>
> flush(stderr()); flush(stdout())
>
> ### Name: word.count
> ### Title: Woerterzaehlfunktion
> ### Aliases: word.count
> ### Keywords: IO
>
> ### ** Examples
>
> ##---- Should be DIRECTLY executable !! ----
> ##-- ==>  Define data, use random,
> ##--    or do  help(data=index)  for the standard data sets.
> word.count('testdatei')
[1] "== word.count startet =="
Warnung in file(file, "r") :
  kann Datei 'testdatei' nicht \"offnen:
  Datei oder Verzeichnis nicht gefunden
Fehler in file(file, "r") : kann Verbindung nicht \"offnen
Calls: word.count -> scan -> file
Ausf\"uhrung angehalten
```

Wir können das Beispiel als nicht ausführbar kennzeichnen oder anders absichern. Wir tauschen die Zeile `word.count("testdatei")` aus:

37 ⟨* 1⟩+ ≡
```
txt<-scan(file=file.path(testpath,"wc","man","word.count.Rd"),"",sep="\n")
ind<-grep("^word.count..testdatei",txt)
if(0<length(ind)){txt<-
     c(txt[1:(ind-1)],
       "# Beispiel 1",
       "\\dontrun{",
       "## Dieses Beispiel geht nur, wenn die Datei testdatei vorliegt.",
       "  word.count('testdatei')",
       "}",
       "# Beispiel 2",
       "if(file.exists('hallo')) word.count('hallo')",
       txt[(ind+1):length(txt)])}
cat(file=file.path(testpath,"wc","man","word.count.Rd"),txt,sep="\n")
"work.count-Help-Seite: Examples verbessert"
```

Neuer Check:

38 ⟨ * 1⟩+ ≡
```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

Jetzt müsste langsam alles klar sein, doch wirft uns noch etwas zurück:

```
...
checking examples ... ERROR
Running examples in 'wc-Ex.R' failed.
The error most likely occurred in:

> ### * wc-package
>
> flush(stderr()); flush(stdout())
>
> ### Name: wc-package
> ### Title: What the package does (short line) ~~ package title ~~
> ### Aliases: wc-package wc
> ### Keywords: package
>
> ### ** Examples
>
> ~~ simple examples of the most important functions ~~
Fehler: Unerwartetes Symbol in "~~ simple examples"
Ausf\"uhrung angehalten
```

2.6.5 Nicht ausführbares Beispiel im Paket-Help-File

`~~ simple examples of the most important functions ~~` führt zu einem Fehler. Eine Suche zeigt, dass diese Zeichenkette in der Paketbeschreibung vorkommt und so syntaktisch nicht korrekt ist. Sie stellt offensichtlich kein ausführbares Beispiel dar. Wir blenden sie einfach per `#` (Kommentarzeichen) aus.

39 ⟨ * 1⟩+ ≡
```
txt<-scan(file=file.path(testpath,"wc","man","wc-package.Rd"),"",sep="\n")
ind<-grep("simple examples of the most important functions",txt)
if(0<length(ind)) txt[ind]<-paste("#",txt[ind])
cat(file=file.path(testpath,"wc","man","wc-package.Rd"),txt,sep="\n")
"Paket-Help-File: Examples auskommentiert"
```

Wir checken noch einmal.

40 ⟨ * 1⟩+ ≡
```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

Und endlich läuft der Prozess durch!

2.7 Dateiinhalte nach der Problembehebung

Wir listen die erarbeiteten Dateien zur Sicherheit noch einmal auf:

2.7.1 Paket-Beschreibung

Der `DESCRIPTION`-File

```
Package: wc
Type: Package
Title: Wortzaehlungen
Version: 0.1
Date: 2008-08-20
Author: P. Wolf
Maintainer: P. Wolf <pwolf@uni.bielefeld.de>
Description: Funktion zur Ermittlung von Haeufigkeiten von Woertern
License: GPL version 2 or newer
URL: http://www.wiwi.uni-bielefelde.de/wolf
LazyLoad: yes
```

2.7.2 Paket-Hilfeseite

Hilfeseite des Pakets:

```
\name{wc-package}
\alias{wc-package}
\alias{wc}
\docType{package}
\title{
What the package does (short line)
~~ package title ~~
}
\description{
More about what it does (maybe more than one line)
~~ A concise (1-5 lines) description of the package ~~
}
\details{
\tabular{ll}{
Package: \tab wc\cr
Type: \tab Package\cr
Version: \tab 1.0\cr
Date: \tab 2008-08-20\cr
License: \tab What license is it under?\cr
LazyLoad: \tab yes\cr
}
~~ An overview of how to use the package, including the most important ~~
~~ functions ~~
}
\author{
Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
~~ The author and/or maintainer of the package ~~
}
\references{
~~ Literature or other references for background information ~~
}
\keyword{ package }
\seealso{
~~ Optional links to other man pages, e.g. ~~
~~ \code{\link[<pkg>:<pkg>-package]{<pkg>}} ~~
}
\examples{
# ~~ simple examples of the most important functions ~~
}
```

2.7.3 Objekt-Hilfeseite

Hilfeseite von `word.count`:

```
\name{word.count}
\alias{word.count}
%- Also NEED an '\alias' for EACH other topic documented here.
\title{ Woerterzaehlfunktion }
\description{
  \code{word.count} zaehlt die Haeufigkeiten von Woertern einer Datei.
}
\usage{
word.count(fname)
}
%- maybe also 'usage' for other objects documented here.
\arguments{
  \item{fname}{ Name der zu untersuchenden Datei }
}
\details{
  \code{word.count} liest den Inhalt der Datei, ermittelt die Woerter und
  zaehlt die Haeufigkeiten.
}
\value{
  Ausgegeben wird ein Vektor der Haeufigkeiten. Die Woerter werden
    als Namen der Objekte verwendet.
```

```
}
\references{
  vgl. Benford's Law: http://en.wikipedia.org/wiki/Benford%27s_law
}
\author{ P. Wolf }

\seealso{ \code{\link{table}}, \code{\link{scan}} }
\examples{
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--    or do  help(data=index)  for the standard data sets.
# Beispiel 1
\dontrun{
## Dieses Beispiel geht nur, wenn die Datei testdatei vorliegt.
  word.count('testdatei')
}
# Beispiel 2
if(file.exists('hallo')) word.count('hallo')
## The function is currently defined as
function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-gsub(" +"," ",txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
  }
}
% Add one or more standard keywords, see file 'KEYWORDS' in the
% R documentation directory.
\keyword{ IO }
```

2.7.4 Erfolgreiches Check-Log

Zum Abschluss sei noch einmal der erfolgreiche Dialog gezeigt.

```
* checking for working pdflatex ... OK
* using log directory '/home/wiwi/pwolf/R/work/test/wc.Rcheck'
* using R version 2.7.0 (2008-04-22)
* using session charset: ISO8859-1
* checking for file 'wc/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'wc' version '0.1'
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking whether package 'wc' can be installed ... OK
* checking package directory ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking for unstated dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
```

```
* checking Rd \usage sections ... OK
* creating wc-Ex.R ... OK
* checking examples ... OK
* creating wc-manual.tex ... OK
* checking wc-manual.tex using pdflatex ... OK
```

Nun schauen wir uns an, welche Dateien im Ergebnisverzeichnis zu finden sind.

41 ⟨* 1⟩+ ≡
```
list.files(file.path(testpath,"wc.Rcheck"),recursive=TRUE)
```

Wir erhalten

```
 [1] "00check.log"             "00install.out"
 [3] "R.css"                   "wc/CONTENTS"
 [5] "wc/DESCRIPTION"          "wc-Ex.ps"
 [7] "wc-Ex.R"                 "wc-Ex.Rout"
 [9] "wc/help/AnIndex"         "wc/help/wc-package"
[11] "wc/help/word.count"      "wc/html/00Index.html"
[13] "wc/html/wc-package.html" "wc/html/word.count.html"
[15] "wc/INDEX"                "wc/latex/wc-package.tex"
[17] "wc/latex/word.count.tex" "wc-manual.aux"
[19] "wc-manual.log"           "wc-manual.out"
[21] "wc-manual.pdf"           "wc-manual.tex"
[23] "wc/man/wc.Rd.gz"         "wc/Meta/hsearch.rds"
[25] "wc/Meta/package.rds"     "wc/Meta/Rd.rds"
[27] "wc/R-ex/wc-package.R"    "wc/R-ex/word.count.R"
[29] "wc/R/wc"                 "wc/R/wc.rdb"
[31] "wc/R/wc.rdx"
```

2.8 Paketbau

Nach erfolgreichem Check können wir endlich das Paket zusammenbauen.

42 ⟨* 1⟩+ ≡
```
setwd(testpath)
system("R CMD build wc")
setwd(wdpath)
```

Wir bekommen die folgenden Meldungen auf der Konsole ausgegeben:

```
* checking for file 'wc/DESCRIPTION' ... OK
* preparing 'wc':
* checking DESCRIPTION meta-information ... OK
* removing junk files
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'wc_0.1.tar.gz'
```

... und im Verzeichnis `test` ist nun das erstellte Paket zu finden: `wc_0.1.tar.gz`. Diese Datei enthält einen gepackten Verzeichnisbaum und damit die Essenz der Arbeit:

```
pwolf@wiwi138:~/R/work/test> ll -R wc
wc:
insgesamt 8
-rw-r--r-- 1 pwolf wiwi 298 22. Aug 09:17 DESCRIPTION
drwxr-xr-x 2 pwolf wiwi 112 22. Aug 09:39 man
drwxr-xr-x 2 pwolf wiwi  80 22. Aug 09:09 R
-rw-r--r-- 1 pwolf wiwi 377 22. Aug 09:09 Read-and-delete-me

wc/man:
insgesamt 8
-rw-r--r-- 1 pwolf wiwi  932 22. Aug 09:25 wc-package.Rd
-rw-r--r-- 1 pwolf wiwi 1517 22. Aug 09:31 word.count.Rd

wc/R:
insgesamt 4
-rw-r--r-- 1 pwolf wiwi 303 22. Aug 09:09 word.count.R
```

Letztlich ist diese Datei `wc_0.1.tar.gz` unser geschaffenes Source-Paket und muss für den Gebrauch nur noch installiert werden.

Es sei erwähnt, dass sich andere auch schon beim Bau von Paketen abgemüht haben. Im Netz findet sich zum Beispiel folgende Seite, der auch einige Bemerkungen zum Troubleshooting zu entnehmen sind: `http://www.maths.bris.ac.uk/~maman/computerstuff/Rhelp/Rpackages.html`

2.9 Paketinstallation

Zunächst benötigen wir ein Bibliotheks-Verzeichnis, in dem wir das Paket unterbringen wollen. Nennen wir es `mylib` und erzeugen es unter `test`.

43 ⟨* 1⟩+ ≡
```
if(!file.exists(libpath)){
  dir.create(libpath)
  cat(libpath,"angelegt\n")
} else {   cat(libpath,"existiert schon\n") }
```

Nun gehen wir in das Verzeichnis und installieren das Paket `wc_0.1.tar.gz`:

44 ⟨* 1⟩+ ≡
```
setwd(testpath)
system(paste("R CMD INSTALL --library=",libpath,
             " wc_0.1.tar.gz",sep=""))
setwd(wdpath)
# entspricht in "test" aus der Linux Konsole dem Befehl:
# R CMD INSTALL --library=/home/wiwi/pwolf/R/work/test/mylib wc_0.1.tar.gz
```

```
* Installing *source* package 'wc' ...
** R
** preparing package for lazy loading
** help
 >>> Building/Updating help pages for package 'wc'
     Formats: text html latex example
  wc-package                      text    html    latex   example
  word.count                      text    html    latex   example
** building package indices ...
* DONE (wc)
```

Jetzt befindet sich in dem Bibliotheks-Verzeichnis `mylib` unser Paket `wc`.

45 ⟨* 1⟩+ ≡
```
system("ls -R test/mylib",TRUE)
# oder aehnlich: list.files(libpath,recursive=TRUE)
```

Nach Aufbereitung bekommen wir:

```
test/mylib:
R.css  wc

test/mylib/wc:
CONTENTS  DESCRIPTION  help  html  INDEX  latex  man  Meta  R  R-ex

test/mylib/wc/help:
AnIndex  wc-package  word.count

test/mylib/wc/html:
00Index.html  wc-package.html  word.count.html

test/mylib/wc/latex:
wc-package.tex  word.count.tex

test/mylib/wc/man:
wc.Rd.gz
```

```
test/mylib/wc/Meta:
hsearch.rds  package.rds  Rd.rds

test/mylib/wc/R:
wc  wc.rdb  wc.rdx

test/mylib/wc/R-ex:
wc-package.R  word.count.R
```

Hinweis: Übrigens lässt sich das Paket auch mit der Funktion `install.packages` installieren. Hierbei wird der Paketname mittels `pkgs` übergeben, die Zielbibliothek durch `lib` und der Ort für Zwischenspeicherung durch `destdir` – also wo im Falle eines Download das nicht installierte Paket abgelegt wird. `repos=NULL` bedeutet, dass kein Download erfolgen, sondern ein lokales Paket Verwendung finden soll.

46 ⟨ * 1⟩+ ≡
```
setwd(testpath)
install.packages(pkgs="wc",lib="mylib",destdir="/tmp",repos=NULL)
setwd(wdpath)
```

2.10 Paketverwendung

Das frisch gebackene Paket können wir jetzt benutzen.

47 ⟨ * 1⟩+ ≡
```
library(wc,lib.loc=libpath)
ls(package:wc)
```

```
/home/wiwi/pwolf/R/work/test/mylib
Thu Aug 21 10:04:45 2008
[1] "word.count"
```

Weiter funktioniert der Zugriff auf die Hilfe:

48 ⟨ * 1⟩+ ≡
```
help(word.count)
```

sowie auch das Anzeigen über die Browser-Hilfe nach:

49 ⟨ * 1⟩+ ≡
```
help.start()
```

Das war's schon.

2.11 Weitergabe von installierten Paketen

Es ist möglich, ein installiertes Paket weiterzugeben. Dazu kann man den Teilbaum des installierten Pakets zippen und so die erneute Installation sparen. Das mag dann empfehlenswert sein, wenn man Kompilationsprozesse nicht auf der Zielmaschine wiederholen will oder kann. Man läuft jedoch bei Verwendung eines anderen Betriebssystems in Gefahr, dass ggf. notwendige Kompilate nicht laufen und Fehler hervorrufen. Diese Pakete nennt man Binärpakete und sind zu erkennen an einer Zeile im `DESCRIPTION`-File, die so aussehen kann:

```
Built: R 2.7.0; ; 2008-08-22 11:51:37; unix
```

Das nicht installierte Paket – `wc_0.1.tar.gz` wird als Source-Paket bezeichnet und enthält keine Zeile mit dem Keyword `Built`.

Zur Demo packen wir einmal unser Werk zusammen.

50 ⟨* 1⟩+ ≡
```
setwd(libpath)
system("zip -r wc_0.1.zip wc/*")
system("mv wc_0.1.zip ..")
setwd(wdpath)
```

Der entstandene Zip-File liegt nun in `test` und kann durch Entpacken in einer anderen Bibliothek an anderer Stelle Verwendung finden.

3 Paktekonstruktion unter Windows

In der Windows-Welt läuft im Prinzip alles entsprechend. Jedoch sind vor dem Paketbau einige Installationen notwendig, da für den Konstruktionsprozess verschiedene Kommandozeilen-Tools, Perl, ein C-Compiler und andere Kleinigkeiten vorhanden sein müssen. Dazu gehören auch LaTeX und *The Microsoft HTML Help Workshop.*

Die meisten notwendigen Tools hat Duncan Murdoch in `Rtools.exe` zusammengebunden, siehe: `http://www.murdoch-sutherland.com/Rtools`.

Im Netz finden wir verschiedene Darstellungen, zum Beispiel eine verlockende unter dem Titel: *Package under windows* – http://faculty.chicagogsb.edu/peter.rossi/research/bayes%20book/bayesm/Making%20R%20Packages%20Under%20Windows.pdf

Die folgenden Abschnitte zeigen einen Erfahrungsbericht zum Paketbau unter Windows.

3.1 Erfolgreicher Strukturentwurf, erfolgloser Paketbau

In Parallelität zur Linux-Lösung starten wir unter Windows unser R (hier durchgeführt mit R-2.7.0) und definieren wie oben die Funktion `word.count`,

51 ⟨* 1⟩+ ≡
```
word.count<-
function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-gsub(" +"," ",txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
}
word.count
```

Wir setzen den Arbeitspfad geeignet (hier: `D:/work`), übernehmen auch die Idee der Pfad-Variablen und legen unter unserer Arbeitsstelle `wdpath` ein Testverzeichnis `test` an.

52 ⟨* 1⟩+ ≡
```
wdpath<-getwd()
testpath<-file.path(wdpath,"test")
libpath<-file.path(wdpath,"test","mylib")

setwd(wdpath)
if(!file.exists(testpath)) dir.create(testpath)
```

In diesem Arbeitsverzeichnis `test` soll das Paket mit dem Namen `wc` entstehen. Nun starten wir die Gerüst-Erzeugungs-Funktion.

53 ⟨* 1⟩+ ≡
```
setwd(wdpath)
package.skeleton(name = "wc", list="word.count",
```

```
                path = "test")
```

Die Erzeugung der Grobkonstruktion funktioniert tadellos.

Paketbau? Mutig starten wir von der R-Konsole den von oben bekannten Befehl.

54 ⟨* 1⟩+ ≡

```
setwd(testpath)
system("R CMD check wc")
setwd(wdpath)
```

Wie erwartet bekommen wir eine Fehlermeldung:

```
Der Befehl "perl" ist entweder falsch geschrieben oder
konnte nicht gefunden werden.
```

Genau das hatten wir erwartet, da ja normalerweise die Rtools nicht da sind. Also beschaffen wir uns diese Tools.

3.2 Installation der erforderlichen Tools

Für die Installation lassen sich die wichtigsten Hinweise der Beschreibung *Writing R Extensions* entnehmen (`http://cran.r-project.org/doc/manuals/R-exts.pdf`) . Weiter helfen `http://www.murdoch-sutherland.com/Rtools/` sowie `http://cran.r-project.org/bin/wi` Auch die Seite `http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset` fasst zusammen, was erforderlich ist.

Aus der letzten Quelle sei ein Stück zitiert:

```
We have collected most of the necessary tools (unfortunately not all, due
to license or size limitations) into an executable installer named
Rtools.exe, available from http://www.murdoch-sutherland.com/Rtools.
You should download and run it, choosing the default Package authoring
installation to build add-on packages, or the full installation if
you intend to build R.
You will need the following items to build R and packages. See the
subsections below for detailed descriptions.
    * Perl (in Rtools.exe)
    * The command line tools (in Rtools.exe)
    * The MinGW compilers (in Rtools.exe)
For building simple packages containing data or R source but no compiled
code, only the first two of these are needed.
A complete build of R including compiled HTML help files and PDF manuals,
and producing the standalone installer R-2.7.1-win32.exe will also need
the following:
    * The Microsoft HTML Help Workshop
    * LaTeX
    * The Inno Setup installer
It is important to set your PATH properly. The Rtools.exe optionally sets
the path to components that it installs.
Your PATH may include . first, then the bin directories of the tools, Perl,
MinGW and LaTeX, as well as the Help Workshop directory. Do not use filepaths
containing spaces: you can always use the short forms (found by dir /x at
the Windows command line). Network shares (with paths starting \\) are not
supported. For example, all on one line,
     PATH=c:\Rtools\bin;c:\Rtools\perl\bin;c:\Rtools\MinGW\bin;c:\texmf\miktex\bin;
          c:\progra~1\htmhe~1;c:\R\bin;c:\windows;c:\windows\system32
It is essential that the directory containing the command line tools comes
first or second in the path: there are typically like-named tools in other
directories, and they will not work. The ordering of the other directories
is less important, but if in doubt, use the order above.

...

E.2 The Microsoft HTML Help Workshop
```

```
To make compiled html (.chm) files you will need the Microsoft HTML Help Workshop,
 currently available for download at msdn.microsoft.com/library/en-us/htmlhelp/html
/hwmicrosofthtmlhelpdownloads.asp and
www.microsoft.com/office/ork/xp/appndx/appa06.htm.
This is not included in Rtools.exe.
```

So ist das also. Untertänig führen wir einen Download der `Rtools` von
`http://www.murdoch-sutherland.com/Rtools/`
durch und von der Seite
`MS-html: http://msdn.microsoft.com/library/`
`default.asp?url=/library/en-us/htmlhelp/html/hwmicrosofthtmlhelpdownloads.asp`
– die auf `http://msdn.microsoft.com/en-us/library/ms669985.aspx` umgelenkt wird – laden wir *The HTML Help Workshop*.

Dann installieren wir zuerst den Workshop und danach die `Rtools` – miktex u.a. befindet sich glücklicherweise schon auf dem Rechner. Durch einen Rechner-Neustart sichern wir ab, dass alles ordentlich eingerichtet ist.

3.3 Der erste richtige Check-Durchlauf

Wir starten nach der Installation der Werkzeuge R und versuchen, wie oben aus der Konsole heraus erfolgreich das Paket zu checken.

55 ⟨* 1⟩+ ≡
```
setwd(testpath)
system("R CMD check wc")
setwd(wdpath)
```

Das Ergebnis ist sehr unbefriedigend, denn der Rechner verharrt in einem ungeklärten Zustand. (Gleiches gilt auch für ein Operieren aus dem `relax`-Editor und auch bei Verwendung von `system("c:/R/R-2.7.0/bin/Rcmd check wc")`.) Bei einem Abbruch durch STOP erkennen wir, dass mitten drin etwas schief gelaufen ist. Auch war festzustellen, dass ein Zugriff auf die Datei `wc.Rcheck/00check.log` unmöglich war, solange nicht die `make`-Prozesse gekillt wurden.

Um alle Synchronisationsprobleme auszuschließen, öffnen wir eine DOS-BOX, gehen zur entsprechenden Stelle und wiederholen unseren Auftrag:

```
c:> D:
d:> cd \work\test
D:\work\test> c:\R\R-2.7.0\bin\Rcmd check wc
```

Der Check läuft erfolgreich durch bzw. es stellen sich dieselben Fehlermeldungen bzgl. der Help-Dateien wie unter Linux ein. Damit haben wir einen Teilerfolg errungen!

3.4 Der erste fehlerfreie Check-Durchlauf

Wir wollen nicht alle kleinen Fehlerbehebungen von oben wiederholen, so dass wir gleich den korrekten `wc`-Teilbaum auf die Windows-Maschine übertragen und den Check wiederholen.

```
D:\work\test> c:\R\R-2.7.0\bin\Rcmd check wc
```

Der Check zeigt uns keine Warnungen oder Fehler an. Also dieses Mal ein voller Erfolg!

3.5 Der Paket-Bau ohne Probleme

Nach soviel Problemen erwarten wir sicher weitere bei der Erzeugung des Pakets. Jedoch ist der Check wirklich das, was man erwartet, denn

```
D:\work\test> c:\R\R-2.7.0\bin\Rcmd build wc
```

läuft durch, und es entsteht der Paket-File: `wc_0.1.tar.gz`

3.6 Die Installation unseres Pakets

Gestärkt durch die Erfolge hoffen wir nun, dass die Installation auch wie von selbst geht. Wir erstellen uns ein spezielles Bibliotheksverzeichnis und aktivieren in der DOS-BOX:

```
D:\work\test> mkdir mylib
D:\work\test> c:\R\R-2.7.0\bin\Rcmd install --library=d:\work\test\mylib wc_0.1.tar.gz
```

Fast! Alles läuft ganz gut, bis auf die winzige Fehlermeldung:

```
hhc: not found
CHM compile failed HTML Help Workshop not installed?
```

In der Tat durch Aktivierung von `"set"` in der DOS-BOX sehen wir, dass es keinen Pfad in das Verzeichnis `HTML Help Workshop` gibt. Goggln ergibt, dass schon andere dieses Problem hatten:

```
Re: R CMD INSTALL error, R-2.7.1
Click to flag this post

by Duncan Murdoch-2 Aug 01, 2008; 08:55pm :: Rate this Message: - Use ratings to moderate (?)

Reply | Reply to Author | Print | View Threaded | Show Only this Message
On 8/1/2008 12:39 PM, Richard Chandler wrote:

> Hi,
>
> I am getting the following error when using R CMD INSTALL ever since I
> upgraded to R-2.7.1:
>
> hhc: not found
> CHM compile failed: HTML Help Workshop not intalled?
>
> As indicated, the package is installed but without CHM help files. I
> have downloaded the latest version of Rtools and I have tried
> uninstalling and reinstalling HTML Help Workshop. I have also tried
> rebuilding the package several times. I was able to build and install
> the package without problems with R-2.7.0.
... [show rest of quote]

I believe that's a warning rather than an error, but what it indicates
is that your hhc help compiler is not on the current PATH.  If you just
type "hhc" at the same command line where you were running "R CMD
INSTALL", you should see

Usage:   hhc <filename>
     where <filename> = an HTML Help project file
Example: hhc myfile.hhp

but I'm guessing you'll see

hhc: not found

To fix this, edit the PATH, and make sure the directory holding hhc.exe
is on it.

Duncan Murdoch
(http://www.nabble.com/R-CMD-INSTALL-error,-R-2.7.1-td18778933.html)
```

Zur Behebung setzen wir den fehlenden Pfad per Hand, das geht mit Hilfe von:

```
Start->Einstellungen->Systemsteuerung->System->erweitert->Umgebungsvariablen
->Systemvariablen->Path-> bearbeiten
```

Einzelne Stellen werden durch `";"` getrennt, jedoch was machen wir mit den Leerzeichen in dem voreingestellten Pfadnamen?

Probeweise installieren wir den Workshop neu, ohne dabei Leerzeichen zu verwenden. Dann ändern wir per Hand den Suchpfad und der Prozess läuft nun durch, sofern wir eine neue DOS-BOX geöffnet haben!

4 Namespaces

4.1 Zweck von Namespaces

Namespaces sind wesentlich dazu da, Objekte eines Paketes in zwei Gruppen aufzuteilen: Die erste Gruppe umfasst die für Anwender bereitgestellten Funktionen und Daten, also die der zu exportierenden Objekten. Die andere Gruppe enthält Objekte die nur innerhalb eines Pakets benötigt werden, nicht jedoch für den allgemeinen Gebrauch vorgesehen sind. Hierdurch kann der Paketentwickler durch Hilfs-Funktionen für sich die Übersicht erhöhen, ohne den Endanwender zu verwirren. Außerdem kann der Entwickler durch genaue Deklaration von zu importierenden Objekten sicher sein, dass er die Funktionen eines bestimmten Pakets zugreift und nicht auf eventuell gleichnamige eines anderen Pakets.

4.2 Definition einer zu verbergenden Hilfsfunktion

Zur Demonstration ändern wir einmal unsere Funktion, indem wir den Teil auslagern, der mehrfache Leerzeichen entfernt. Für diese Aufgabe schreiben wir die Funktion `.entferne.mehrfache.`
Der Name beginnt absichtlich mit `"i."` gemäß der Idee, die Namen aller nicht zu exportierenden Objekte mit `"i."` für *intern* beginnen lassen. Auf diese Weise kann der Entwickler sofort erkennen, ob ein Objekt exportiert wird. Das muss man nicht auf diese Weise machen, der Vorschlag weist aber darauf hin, sich gleich zu Beginn zweckmäßige Designregeln aufzuerlegen. Hier die Umsetzung.

56 ⟨* 1⟩+ ≡

```
word.count<-function(fname){
  print("== word.count startet ==")
  # Inputcheck
  # Funktionen fixieren
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-i.entferne.mehrfache.Leerzeichen(txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  return(res)
}
dump("word.count",
     file=file.path(testpath,"wc/R/word.count.R"))
i.entferne.mehrfache.Leerzeichen<-function(txt){
  txt<-gsub(" +"," ",txt)
  txt
}
dump("i.entferne.mehrfache.Leerzeichen",
     file=file.path(testpath,"wc/R/interneFns.R"))
prompt(i.entferne.mehrfache.Leerzeichen)
```

Ein Check-Versuch müssten anzeigen, dass die Hilfeseite noch fehlt. Deshalb entwerfen sofort eine kurze Seite:

57 ⟨* 1⟩+ ≡
```
txt<-"\\name{i.entferne.mehrfache.Leerzeichen}
\\alias{i.entferne.mehrfache.Leerzeichen}
\\title{ Leerzeichenentfernung }
\\description{ siehe Titel }
\\usage{ i.entferne.mehrfache.Leerzeichen(txt) }
\\arguments{
  \\item{txt}{ ~~Describe \code{txt} here~~ }
}
\\details{ interne Funktion des Pakets }
\\value{ Text ohne mehrfache Leerzeichen }
\\author{ pw }
\\examples{
## The function is currently defined as
}"
cat(txt,file=file.path(testpath,"wc","man","interneFNS.Rd"))
"Rd-File geschrieben"
```
und checken erfolgreich:

58 ⟨* 1⟩+ ≡
```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

4.3 Festlegung der zu exportierenden Objekte

Die Objekte, die exportiert oder importiert werden sollen, werden in einer Datei mit dem Namen `NAMESPACE` festgelegt. Zur Illustration werfen wir einen Blick in die Deklarationsdatei des Pakets `boot`:

59 ⟨* 1⟩+ ≡
```
cat(readLines(file.path(R.home(),"library","boot","NAMESPACE")),sep="\n")
```

Hier ist der Dateiinhalt:

```
export(abc.ci, boot, boot.array, boot.ci, censboot, control, corr,
       cum3, cv.glm, EEF.profile, EL.profile, empinf, envelope,
       exp.tilt, freq.array, glm.diag, glm.diag.plots, imp.moments,
       imp.prob, imp.quantile, imp.weights, inv.logit,
       jack.after.boot, k3.linear, lik.CI, linear.approx, logit,
       nested.corr, norm.ci, saddle, saddle.distn, simplex,
       smooth.f, tilt.boot, tsboot, var.linear)
# documented but not exported
# export(lines.saddle.distn, plot.boot, print.boot, print.bootci, print.simplex)
importFrom(graphics, lines, plot)
S3method(lines, saddle.distn)
S3method(plot, boot)
S3method(print, boot)
S3method(print, bootci)
S3method(print, saddle.distn)
S3method(print, simplex)
```

Die für uns zentrale Anweisung lautet `export`. In dieser sind alle Objekte aufgezählt, die exportiert werden sollen. Weiter werden aus dem Paket `graphics` die Funktionen `lines` und `plot` benötigt. Damit kann eine anderswo definierte Funktion `plot` nicht für Probleme sorgen. Zum Schluss werden noch verschiedene S3-Methoden *registriert*, was wir hier nicht weiter kommentieren.

Also schreiben wir entsprechend eine Namespace-Datei für unser Paket.

60 ⟨* 1⟩+ ≡
```
txt<-"export(word.count)"
cat(txt,"\n",file=file.path(testpath,"wc","NAMESPACE"))
"NAMESPACE-Datei geschrieben"
```

Das war es schon. Falls wir die Idee aufgreifen wollen, alle Funktionen zu exportieren, die nicht mit "i." beginnen, hätten wir die zu exportierenden Funktionen auch mit einem regulären Ausdruck beschreiben können. Dann hätten wir als Exportanweisungen schreiben können:

61 ⟨* 1⟩+ ≡
```
txt<-c(
# alle Funktionen, die nicht mit "i" beginnen
  'exportPattern("^[^i]")',
# sowie alle, die mit "i" beginnen, aber dann keinen Punkt haben
  'exportPattern("^[i][^\\\\.]")'
)
```

Wir wollen mal schauen, ob der Entwurf funktioniert.

62 ⟨* 1⟩+ ≡
```
setwd(testpath);system("R CMD check wc");setwd(wdpath)
```

Dieser Schritt funktioniert einwandfrei und die nächsten folgen sogleich:

63 ⟨* 1⟩+ ≡
```
setwd(testpath)
system("R CMD build wc")
system(paste("R CMD INSTALL --library=",libpath,
             " wc_0.1.tar.gz",sep=""))
setwd(wdpath)
```

Fertig.

4.4 Einsatztest

Jetzt wollen wir mal sehe, ob der Export klappt.

64 ⟨* 1⟩+ ≡
```
setwd(wdpath)
if(length(grep("package:wc",search()))>0) detach("package:wc")
library(wc,lib.loc=libpath)
```

```
 [1] "wc"        "relax"     "tcltk"     "stats"     "graphics"  "grDevices"
 [7] "utils"     "datasets"  "methods"   "base"
```

Wie sieht die Funktionsübersicht aus?

65 ⟨* 1⟩+ ≡
```
ls(pos=2)
```

```
[1] "word.count"
```

Und die Funktion selbst?

66 ⟨* 1⟩+ ≡
```
wc::word.count
```

```
function (fname)
{
    print("== word.count startet ==")
    txt <- scan(fname, "", sep = "")
    txt <- gsub("[^a-zA-Z]", " ", txt)
    txt <- i.entferne.mehrfache.Leerzeichen(txt)
    txt <- unlist(strsplit(txt, " "))
    res <- rev(sort(table(txt)))
    cat("almost completed\n")
    return(res)
}
<environment: namespace:wc>
```

Ist die interne Funktion erreichbar?

67 ⟨* 1⟩+ ≡
```
wc::i.entferne.mehrfache.Leerzeichen
```

Wir erhalten die erhoffte Fehlermeldung:

```
Error : 'i.entferne.mehrfache.Leerzeichen' ist kein von 'namespace:wc' exportiertes Objekt
```

Jedoch führt ein harter Zugriff zum Erfolg:

68 ⟨* 1⟩+ ≡
```
wc:::i.entferne.mehrfache.Leerzeichen
```

Hier ist die Definition der verborgenen Funktion!

```
function (txt)
{
    txt <- gsub(" +", " ", txt)
    txt
}
<environment: namespace:wc>
```

Damit ist grob die Funktionsweise eines `NAMESPACE` an einem winzigen Beispiel demonstriert. Bei größeren wird es schon entsprechend gehen.

5 C-Programme

Was sehr langsam geht, kann manchmal durch Einsatz eines C-Programms viel schneller gemacht werden. Im Folgenden werden zwei kleine Demonstrationen vorgeführt. Der nächste Abschnitt zeigt, wie man eine übersetzte C-Programm einbindet. Danach wollen wir die C-Lösung in unser Paket aufnehmen. Für Vergleichbarkeit stützt sich das gewählte Beispiel auf Ligges (2004). Wir schreiben damit eine Funktion, die zwei Vektoren addiert und das Additionsergebnis zurückliefert.

5.1 C-Programm-Einbindung in R

Folgende Schritte sind zu erledigen:

1. definiere C-Programm
2. übersetze C-Programm
3. lade (dynamisch) erzeugtes Objekt-Datei
4. rufe Programm auf
5. entlade Objekte-Datei

Das C-Programm. Im ersten Schritt ist ein C-Programm zu schreiben.

69 ⟨* 1⟩+ ≡
```
prog.c<-
"
#include <Rinternals.h>
SEXP addiere(SEXP a, SEXP b)
{
   int i, n;
   n = length(a);
   for(i=0;i<n;i++){
      /* asdf */
```

```
      REAL(a)[i] += REAL(b)[i];
    }
    return(a);
}
"
cat(file=file.path(testpath,"prog.c"),prog.c,"\n")
"C-Programm erstellt"
```

Compilation. Zweitens ist das C-Programm zu übersetzen.

70 ⟨ * 1⟩+ ≡
```
setwd(testpath)
system("R CMD SHLIB prog.c",TRUE)
setwd(wdpath)
```

Als Ausgabe erhalten wir:

```
[1] "gcc -std=gnu99 -I/usr/lib/R/include  -I/usr/local/include    -fpic  -g -O2 -c prog.c -o prog.o"
[2] "gcc -std=gnu99 -shared -L/usr/local/lib -o prog.so prog.o   "
```

Laden. Im dritten Schritt laden wir dynamisch die erzeugte Datei (`DLL` bzw. `so`-Datei) Objekt, ggf. mit vorherigem Entladen.

71 ⟨ * 1⟩+ ≡
```
setwd(testpath)
try(dyn.unload("prog.so"))
dyn.load("prog.so")
setwd(wdpath)
```

Ausgabe:

```
DLL name: prog
Filename: /home/wiwi/pwolf/R/work/test/prog.so
Dynamic lookup: TRUE
```

Wir wollen mal sehen, welche Dateien wir durch die Compilation bekommen haben:

72 ⟨ * 1⟩+ ≡
```
setwd(wdpath)
grep("^prog",list.files(testpath),value=TRUE)
```

```
[1] "prog.c"  "prog.o"  "prog.so"
```

Verwendung. Nun können wir das Programm verwenden.

73 ⟨ * 1⟩+ ≡
```
n<-1000000
a<-1:n+0.1
b<-rnorm(n)
print(system.time({a+b}))
print(system.time({.Call("addiere", as.double(a), as.double(b))}))
"ok"
```

Leider zeigt sich bei diesem Beispiel kein Spareffekt.

```
   User      System verstrichen
  0.000       0.012       0.014
   User      System verstrichen
  0.016       0.000       0.016
[1] "ok"
```

Entladen. Der Ordnung halber entfernen wir die geladene Objekt wieder.

74 ⟨* 1⟩+ ≡
```
dyn.unload("prog.so")
```

Soviel zum einfachen C-Beispiel.

5.2 C-Compilate in Paketen

Jetzt wollen wir natürlich ein C-Programm innerhalb einer Funktion unseres Pakets einsetzen. Das erfordert keine großen Veränderungen, denn der Compilationsprozess wird während der Installation eines Pakets automatisch durchgeführt. Als Experiment bauen ohne Frage nach dem Sinn das C-Programm aus dem Ligges-Buch in unsere `word.count`-Funktion ein. Im folgenden Abschnitt modifizieren wir die Lösung so, dass der Aufruf des Objekt-File nicht völlig bedeutungslos ist.

Zunächst müssen wir den Code in einem neu zu schaffenden Verzeichnis mit dem Namen `src` unterzubringen. Also gilt es: Verzeichnis anlegen ...

75 ⟨* 1⟩+ ≡
```
setwd(testpath)
if(!file.exists(file.path("wc","src")))
  dir.create(file.path("wc","src"))
setwd(wdpath)
```

... und R-Funktion sowie C-Programm ablegen.

76 ⟨*speichere Programme* 76⟩ ≡
```
prog.c<-"
#include <Rinternals.h>
SEXP addiere(SEXP a, SEXP b)
{
   int i, n;
   n = length(a);
   for(i=0;i<n;i++){ REAL(a)[i] += REAL(b)[i];    }
   return(a);
}
"
cat(file=file.path(testpath,"wc","src","addprog.c"),prog.c,"\n")
cat("C-Programm in src abgelegt")
word.count<-function(fname){
  print("== word.count startet ==")
  txt<-scan(fname,"",sep="")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-i.entferne.mehrfache.Leerzeichen(txt)
  txt<-unlist(strsplit(txt," "))
  res<-rev(sort(table(txt)))
  cat("almost completed\n")
  cat("Anzahl:")
  b<-a<-1:5
  aa<- .Call("addiere", as.double(a), as.double(b),PACKAGE="wc")
  print(aa)
  return(res)
}
dump("word.count",
     file=file.path(testpath,"wc/R/word.count.R"))
```

Dann kannt der Check wie gewohnt ablaufen ...

77 ⟨*checke* `wc` 77⟩ ≡
```
setwd(testpath)
try(system("rm /home/wiwi/pwolf/R/work/test/wc/src/*.c"))
try(system("rm /home/wiwi/pwolf/R/work/test/wc/src/*.o"))
try(system("rm /home/wiwi/pwolf/R/work/test/mylib/libs/*.o"))
```
⟨*speichere Programme* 76⟩

```
system("R CMD check wc")
setwd(wdpath)
```

... und ebenso Bau und Installation:

78 ⟨*baue und installiere* `wc` 78⟩ ≡

```
setwd(testpath)
system("R CMD build wc")
system(paste("R CMD INSTALL --library=",libpath,
             " wc_0.1.tar.gz",sep=""))
setwd(wdpath)
```

Hiernach findet man in dem Verzeichnis `mylib/wc/libs` die Datei `wc.so`, die unsere compilierte Funktion enthalten müsste. Wir machen die Nagelprobe, etwa so:

79 ⟨* 1⟩+ ≡

```
setwd(wdpath)
if(length(grep("package:wc",search()))>0) detach("package:wc")
library(wc,lib.loc="test/mylib")
word.count("c")[1:3]
```

Es folgt:

```
[1] "== word.count startet =="
Read 7699 items
almost completed
Anzahl:Fehler in .Call("addiere", as.double(a), as.double(b), PACKAGE = "wc") :
  C Symbolname "addiere" nicht in der DLL f\"ur Paket "wc"
```

Leider kommt es zu einer Fehlermeldung. `addiere` wird offensichtlich nicht gefunden. Entweder ist die `so`-Datei defekt, oder irgend etwas ist nicht korrekt eingebunden. Wenn wir die `so`-Datei per Hand laden, wird unser compiliertes C-Programm erfolgreich gestartet:

80 ⟨* 1⟩+ ≡

```
dyn.load("mylib/wc/libs/wc.so")
word.count("c")[1:3]
```

```
[1] "== word.count startet =="
Read 7699 items
almost completed
Anzahl:[1]  2.6  5.2  7.8 10.4 13.0
txt
       wc    R
1691  189  180
```

Wir müssen also nur noch erreichen, dass das Shared-Objekt beim Öffnen des Pakets gleich mit geladen wird. In dem Erweiterungs-Manual lesen wir:

> *Various R functions in a package can be used to initialize and clean up. For packages without a name space, these are* `.First.lib` *and* `.Last.lib`. *(See Section 1.6.3 [Load hooks], page 21, for packages with a name space.) It is conventional to define these functions in a file called* `?zzz.R?`. *If* `.First.lib` *is defined in a package, it is called with arguments* `libname` *and* `pkgname` *after the package is loaded and attached. (If a package is installed with version information, the package name includes the version information, e.g.* `?ash_1.0.9?`*.) A common use is to call* `library.dynam` *inside* `.First.lib` *to load compiled code: another use is to call those functions with side effects. If* `.Last.lib` *exists in a package it is called (with argument the full path to the installed package) just before the package is detached. It is uncommon to detach packages and rare to have a* `.Last.lib` *function: one use is to call* `library.dynam.unload` *to unload compiled code.*[1]

Weiter erfahren wir:

[1] Writing R Extensions, Version 2.7.1 (2008-06-23), Kap. 1.1.3 Package directories, S. 6

Packages with name spaces do not use the .First.lib function. Since loading and attaching are distinct operations when a name space is used, separate hooks are provided for each. These hook functions are called .onLoad and .onAttach. They take the same arguments as .First.lib; they should be defined in the name space but not exported.[2]

Geeignet zur Lösung dieses Problems ist also im Falle ohne Namespace die Funktion `.First.lib`. Diese wird beim Öffnen eines Pakets vorweg ausgeführt. Bei der Verwendung des Namespace-Konzepts ist dagegen `.onLoad` einzusetzen. Beim Abhängen eines Pakets können mit der Funktion `.onUnload` Säuberungsprozesse aktiviert werden bzw. mit `.Last.lib` für den Fall ohne Namespaces.

Also entwerfen wir `zzz.R` und starten erneut unsere Prozedur:

81 ⟨*zzz.R* 81⟩ ≡
```
txt<-'
.onLoad <- function(libname, pkgname)
{
    library.dynam("wc", pkgname, libname)
}
.onUnload <- function(libpath)
    library.dynam.unload("wc", libpath)
'
cat(txt,file=file.path(testpath,"wc","R","zzz.R"))
cat("zzz.R angelegt")
```
⟨*checke* `wc` 77⟩
⟨*baue und installiere* `wc` 78⟩

Nach Aktivierung der Testanweisungen

82 ⟨* 1⟩+ ≡
```
setwd(wdpath)
if(length(grep("package:wc",search()))>0) detach("package:wc")
library(wc,lib.loc="test/mylib")
word.count("c")[1]
```
... stellt sich nun der Erfolg direkt ein:

```
[1] "== word.count startet =="
Read 7699 items
almost completed
Anzahl:[1]  2.6  5.2  7.8 10.4 13.0

1691
```

Was wollen wir noch mehr? Vielleicht eine

5.3 Letzte Demo

Zum Schluss wollen wir unsere `word.count`-Lösung so ändern, dass in dem C-Prgramm die relative Anzahl der Wörter bestimmt und ausgegeben werden.

Sicherheitshalber entfernen wir alle alten Objekte.

83 ⟨*clean: entferne alte Compilate* 83⟩ ≡
```
try(system("rm /home/wiwi/pwolf/R/work/test/wc/src/*.o"))
try(system("rm /home/wiwi/pwolf/R/work/test/mylib/libs/*.o"))
try(system("rm /home/wiwi/pwolf/R/work/test/wc/src/*.c"))
```

und definieren neu:

84 ⟨*definiere Programme* 84⟩ ≡
```
setwd(testpath)
prog.c<-
```

[2] Writing R Extensions, Version 2.7.1 (2008-06-23), Kap. 1.6.3 Load hooks, S. 21

```
"
#include <Rinternals.h>
SEXP anteil(SEXP a)
{
   int i, n;
   double b;
   b = 0;
   n = length(a);
   for(i=0;i<n;i++){
      b += REAL(a)[i];
   }
   for(i=0;i<n;i++){
      REAL(a)[i] = REAL(a)[i]/b;
   }
   return(a);
}
"
cat(file=file.path(testpath,"wc","src","anteil.c"),prog.c,"\n")
"C-Programm prog.c in src abgelegt"

word.count<-function(fname){
  print("== word.count startet ==")
  txt<-scan(fname,"",sep="\n")
  txt<-gsub("[^a-zA-Z]"," ",txt)
  txt<-i.entferne.mehrfache.Leerzeichen(txt)
  txt<-unlist(strsplit(txt," "))
  txt<-txt[txt!=" "]; txt<-txt[txt!=""]
  res<-rev(sort(table(txt)))
  anteil<-.Call("anteil", as.double(res),PACKAGE="wc")
  plot(anteil,type="h",log="x")
  res<-cbind(res,anteil)
}
dump("word.count",
     file=file.path(testpath,"wc/R/word.count.R"))
```

Jetzt kann die Produktion beginnen!

85 ⟨*produziere Paket* 85⟩ ≡

```
setwd(testpath)
system("R CMD check wc")
system("R CMD build wc")
system(paste("R CMD INSTALL --library=",libpath,
             " wc_0.1.tar.gz",sep=""))
setwd(wdpath)
```

Ein Test zeigt, dass die Modifikation die gewünschten Früchte trägt.

5.4 Nachbemerkung

Für das gestellte Problem haben wir einen Weg aufgezeigt aufgezeigt, um auch noch ein übersetztes Programm einzubingen. In den Beschreibungen ist zu finden, dass man bei der Verwendung des Namespace-Konzepts auch mit einer Namespace-Deklaration shared objects bekannt machen können. Dazu müsste als Ergänzung ein Eintrag in der Form `useDynLoad(wc)` in der Namespace-Datei `NAMESPACE` vorgenommen werden.

86 ⟨ * 1⟩+ ≡

```
txt<-
"
useDynLoad(wc)
export(word.count)
"
cat(txt,"\n",file=file.path(testpath,"wc","NAMESPACE"))
```

"NAMESPACE-Datei mit useDynLoad geschrieben"
Unter den vorhandenen Testbedingungen waren die durchgeführten Versuche jedoch nicht erfolgreich, so dass wir an dieser Stelle die vorgestellte, explizite Ladeoperation empfehlen.

6 Upload nach CRAN

Für den Upload eines Pakets auf den CRAN-Server sind drei Dinge zu berücksichtigen.

1. entwickle das Paket soweit, dass es aus eigener Sicht fertig ist und nicht nur ein Entwurf
2. bearbeite das Paket so lange, bis keine Fehlermeldungen oder (schweren) Warnungen mehr auftauchen.
3. lade das Paket hoch nach `cran.R-project.org/incoming/`

   ```
   ftp ftp://cran.R-project.org/incoming/
   ftp> put wc_0.1.tar.gz
   ftp> bye
   ```

   und schicke eine Begleitmail an `cran@r-project.org`.

Falls alles perfekt gestaltet ist, wird das Paket in wenigen Tagen der Welt zur Verfügung stehen.

7 Literatur

Als Start für das Literaturstudium werden nur wenige Quellen genannt. Weitere lassen sich über die R-Projektseite `http://cran.r-project.org/` finden.

H. P. Wolf (2004): Kleiner R-Steilkurs,
`http://www.wiwi.uni-bielefeld.de/~wolf/lehre/ss04/liptex/Rkurs.pdf`

H. P. Wolf (2004): Funktionsdefinitionen in R — diskutiert am Beispiel: Funktionsschnittpunkte, `http://www2.wiwi.uni-bielefeld.de/~wolf/lehre/ss04/liptex/fnsinR.pdf`

W. N. Venables, D. M. Smith and the R Development Core Team (2008): An Introduction to R, `http://cran.r-project.org/doc/manuals/R-intro.pdf`

U. Ligges (2004): Programmieren mit R, Springer, Berlin

R Development Core Team (2008): Writing R Extensions,
`http://cran.r-project.org/doc/manuals/R-exts.pdf`

Weiter wurden noch erwähnt:

`http://www.maths.bris.ac.uk/~maman/computerstuff/Rhelp/Rpackages.html`

`http://www.murdoch-sutherland.com/Rtools`.

`http://faculty.chicagogsb.edu/peter.rossi/research/bayes%20book/bayesm/Making%20R%20Packages%20Under%20Windows.pdf`

`http://cran.r-project.org/bin/windows/base/rw-FAQ.html`

`http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset`

`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwmicrosofthtmlhelpdownloads.asp`

`http://msdn.microsoft.com/en-us/library/ms669985.aspx` umgelenkt wird –

`http://www.nabble.com/R-CMD-INSTALL-error,-R-2.7.1-td18778933.html`