

# R Pakete entwickeln

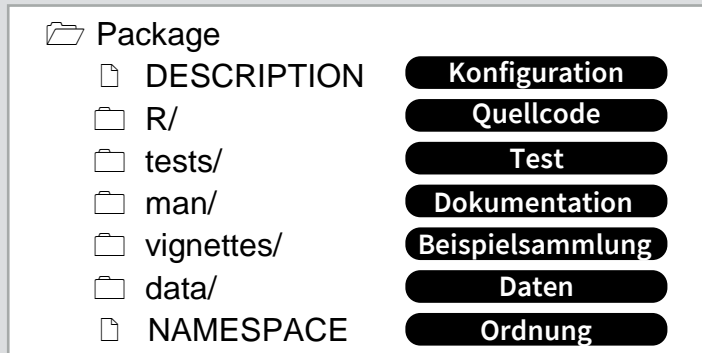
mit devtools Schummelzettel



## Struktur

Ein Paket ist eine Vereinbarung zum Organisieren von Dateien in Verzeichnissen.

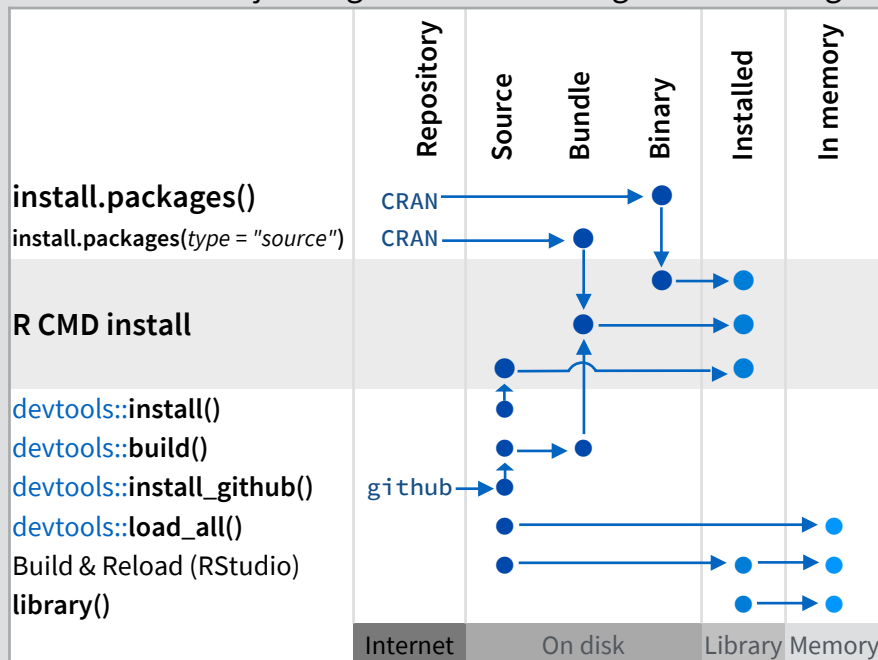
Hier wird illustriert, wie die 7 häufigsten Bestandteile eines Software-Paketes in R verwendet werden:



Der Inhalt eines Pakets kann gespeichert werden als:

- **Quellcode** (engl. **source**) - ein Ordner mit Unterverzeichnissen (siehe oberhalb)
- **Bündel** (engl. **bundle**) - eine komprimierte Datei (.tar.gz)
- **Binärdatei** (engl. **binary**) - eine komprimierte Datei, optimiert für ein bestimmtes Betriebssystem

Oder in einer Programmbibliothek installiert (und während einer R-Sitzung in den Arbeitsspeicher geladen) oder online in einem Repository archiviert. Die folgenden Funktionen beschreiben den jeweiligen Status und mögliche Änderungen:



`devtools::add_build_ignore("Datei")`

Fügt *Datei* zu `.Rbuildignore` hinzu. Dies ist eine Liste von Dateien, die nicht inkludiert werden, wenn das Paket im Build erstellt wird.

## Konfiguration (DESCRIPTION)

Die Datei `DESCRIPTION` beschreibt dieses Paket und wie es mit anderen Paketen zusammenspielt.

- ✓ Die Datei namens `DESCRIPTION` muss existieren
- ✓ Die Pakete, von denen dieses Paket abhängt, hinzufügen mit `devtools::use_package()`  
Fügt ein Paket zum „Imports“-Feld (oder „Suggests“-Feld hinzu (falls das 2. Argument „Suggests“ ist).

### CCO

Ohne weitere Bedingungen.

### MIT

MIT Lizenz trifft auf diesen Quellcode zu, wenn er weiter geteilt wird.

### GPL-2

GPL-2 Lizenz trifft auf diesen Quellcode zu und auf jeden Code, mit dem er gebündelt wird, wenn er weiter geteilt wird.

```
Package: meinPaket
Title: Titel des Paketes
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
  "hadley@me.com", role = c("aut", "cre"))
Description: Zweck des Paketes (ein Paragraph)
Depends: R (>= 3.1.0)
License: GPL-2
LazyData: true
Imports:
  dplyr (>= 0.4.0),
  ggvis (>= 0.2)
Suggests:
  knitr (>= 0.1.0)
```

**Importiert** Pakete ohne die dieses Paket nicht funktioniert. R installiert sie zusammen mit diesem Paket.

**Empfiehl** Pakete die nicht zwingend notwendig sind für dieses Paket. Benutzer können sie manuell installieren.

## Quellcode (R/)

Der gesamte R-Quellcode wird im Ordner `R/` gespeichert. Ein Paket mit nur diesem Ordner `R/` ist trotzdem nützlich.

- ✓ Neues Paket/Projekt anlegen mit `devtools::create("Pfad/zum/Namen")`  
Erstellt eine Vorlage um ein Paket zu entwickeln.
- ✓ Quellcode in `R/` als Script speichern (mit Dateierdung `.R`)

## Arbeitsfluss

1. Bearbeiten des Quellcodes.
2. Code laden mit einem der folgenden Befehle `devtools::load_all()`  
Lädt alle *gespeicherten* Dateien in `R/` in den Speicher.  
**Strg/Cmd + Shift + L (Tastaturkürzel)**  
Speichert alle offenen Dateien und führt `load_all()` aus.
3. Experimentieren in der Konsole.
4. Wiederholen.

- Konsistenten Stil verwenden, siehe [r-pkgs.had.co.nz/r.html#style](http://r-pkgs.had.co.nz/r.html#style)
- Funktion markieren und **F2** drücken um die Definition zu sehen
- Suche nach Funktionen mittels **Strg + .**

## Mehr Informationen auf [r-pkgs.had.co.nz](http://r-pkgs.had.co.nz)

Webseite <http://r-pkgs.had.co.nz> • devtools 1.6.1 • Update: 1/15  
RStudio® ist ein eingetragenes Markenzeichen von RStudio, Inc.  
All rights reserved • [info@rstudio.com](mailto:info@rstudio.com) • 844-448-1212 • [rstudio.com](http://rstudio.com)

## Test (tests/)

Der Ordner `tests/` enthält Modultests/Unittests um Fehler in der Funktionalität zu finden.

- ✓ Dateiverzeichnis `tests/` erstellen und `testthat` importieren mit `devtools::use_testthat()`  
Richtet dieses Paket so ein, dass `testthat` für automatische Tests verwendet wird.
- ✓ Tests und erwartete Verhaltensweisen mit `context()` und `test()` festlegen
- ✓ Tests als `.R` Dateien in `tests/testthat/` speichern

### Arbeitsfluss

1. Code oder Tests bearbeiten.
2. Code testen mit einem von `devtools::test()`  
Führt alle Tests aus die in `tests/` gespeichert sind.  
**Strg/Cmd + Shift + T (Tastaturkürzel)**
3. Wiederholen bis alle Tests bestanden werden.

### Beispiel-Test

```
context("Arithmetik")

test_that(„Mathe is ok“, {
  expect_equal(1 + 1, 2)
  expect_equal(1 + 2, 3)
  expect_equal(1 + 3, 4)
})
```


<code>expect_equal()</code>	ist bis auf kleine numerische Abweichung gleich?
<code>expect_identical()</code>	ist gleich (exakt)?
<code>expect_match()</code>	entspricht festgelegter Zeichenfolge /regulärem Ausdruck?
<code>expect_output()</code>	zeigt vorgegebene Ausgabe an?
<code>expect_message()</code>	zeigt festgelegte Mitteilung an?
<code>expect_warning()</code>	zeigt festgelegte Warnung an?
<code>expect_error()</code>	gibt festgelegte Fehlermeldung zurück?
<code>expect_is()</code>	Ausgabe vererbt von einer bestimmten Klasse?
<code>expect_false()</code>	Rückgabewert ist FALSE?
<code>expect_true()</code>	Rückgabewert ist TRUE?

## Dokumentation ( man/)

Der Ordner  man/ enthält Hilfedateien und eine Dokumentation der Funktionen in diesem Paket.

- roxygen** Kommentare verwenden um jede Funktion direkt bei ihrer Definition zu dokumentieren
- Namen jedes exportierten Datensatzes dokumentieren
- Hilfreiche Beispiele für jede Funktion liefern

### Arbeitsfluss

1. Einfügen von **roxygen** Kommentaren in .R Dateien.
2. Umwandeln von **roxygen** Kommentare in Dokumentation mit einem der folgenden Befehle **devtools::document()**  
Wandelt roxygen Kommentare in .Rd Dateien um und speichert sie in  man/. Baut auch den **NAMESPACE** auf.  
**Strg/Cmd + Shift + D** (Tastaturkürzel)
3. Öffnen der Vorschau von Hilfedateien der Dokumentation mittels **?**.
4. Wiederholen.

### Kennzeichen zur .Rd Formattierung

<code>\emph{}</code>	<code>\code{\link{}}</code>	<code>\dontrun{}</code>
<code>\strong{}</code>	<code>\link{}</code>	<code>\dontshow{}</code>
<code>\code{}</code>	<code>\link[package]{}</code>	<code>\donttest{}</code>
<code>\pkg{}</code>	<code>\linkS4class{}</code>	
<code>\email{}</code>	<code>\deqn{}</code>	<code>\tabular{lcr}</code>
<code>\href{}</code>	<code>\eqn{}</code>	<code>\tab</code>
<code>\url{}</code>		<code>\cr</code>

## Das Paket „roxygen“

**roxygen** erlaubt das Schreiben von Dokumentation innerhalb der .R Dateien mit einer speziellen Syntax.


- **roxygen** Dokumentation wird aus Kommentarzeilen mit vorangestelltem **#'** erstellt.
- Diese Kommentarzeilen werden direkt über halb des Codes, der die Funktion definiert, angelegt.
- Ein **roxygen @** Kennzeichen wird rechts neben **#'** eingefügt um einen bestimmten Abschnitt der Dokumentation zu beliefern.
- Kommentarzeilen ohne Kennzeichen werden für Titel-, Beschreibung - und Detail-Abschnitt verwendet (in dieser Reihenfolge).


```
#' Zwei Zahlen addieren.  
#'  
#' @param x Eine Zahl.  
#' @param y Eine Zahl.  
#' @return Die Summe von {x} und {y}.  
#' @examples  
#' add(1, 1)  
#' @export  
add <- function(x, y) {  
  x + y  
}
```

### Häufige roxygen Kennzeichen

@aliases	@inheritParams	@seealso
@concepts	@keywords	@format
@describeIn	@param	@source Daten
@examples	@rdname	@include
@export	@return	@slot S4
@family	@section	@field RC


## Beispielsammlung ( vignettes/)

Der Ordner  vignettes/ enthält Dokumente die den Benutzern zeigen, wie Problemstellungen mit diesem Paket gelöst werden können.

- Ordner  vignettes/ anlegen und Mustervorlage einer Vignette erstellen mit **devtools::use\_vignette()**  
Erstellt Vorlage als vignettes/my-vignette.Rmd.
- YAML Kopfzeile an die Vignette anhängen (siehe rechts)
- Den Inhalt der Vignette in R Markdown erstellen ([rmarkdown.rstudio.com](http://rmarkdown.rstudio.com))

```
---  
title: "Vignetten-Titel"  
author: "Vignetten-Autor"  
date: "`r Sys.Date()`"  
output: rmarkdown::html_vignette  
vignette: >  
  %\VignetteIndexEntry{Vignette Title}  
  %\VignetteEngine{knitr::rmarkdown}  
  \usepackage[utf8]{inputenc}  
---
```

## Daten ( data/)

Der Ordner  data/ erlaubt das Einfügen von Datensätzen im Paket.

- Daten speichern in **data/**, **R/Sysdata.rda** oder **inst/extdata**
- LazyData: true** muss in der Datei **DESCRIPTION** verwendet werden
- Daten als **.Rdata** speichern (empfohlen)

### devtools::use\_data()

Erstellt ein Datenobjekt in **data/** (oder in **R/Sysdata.rda** falls **internal = TRUE**).


### devtools::use\_data\_raw()

Erstellt ein R Script in **data-raw/** um einen Datensatz zu bereinigen. Inkludiert **data-raw/** in **.Rbuildignore**.

Daten werden gespeichert in

- **data/** um sie allen Benutzern des Pakets zugänglich zu machen
- **R/sysdata.rda** um interne Daten für die entwickelten Funktionen zu hinterlegen.
- **inst/extdata** um nicht bereinigte Daten verfügbar zu machen zum Laden und Parsen von Beispielen. Zugriff auf die Daten erfolgt mittels **system.file()**

## Ordnung ( NAMESPACE)

Die Datei  NAMESPACE hilft dabei, dieses Paket selbstständig zu machen: es beeinträchtigt kein anderes Paket und umgekehrt.

- Funktionen für Benutzer exportieren indem **@export** in ihre **roxygen** Kommentare eingefügt wird
- Objekte von anderen Paketen importieren mit **package::object** (empfohlen) oder **@import**, **@importFrom**, **@importClassesFrom**, **@importMethodsFrom** (nicht immer empfohlen)

### Arbeitsfluss

1. Quellcode oder Tests bearbeiten.
2. Paket dokumentieren (**devtools::document()**).
3. **NAMESPACE** überprüfen.
4. Wiederholen bis **NAMESPACE** korrekt ist.

## Fertiges Paket einreichen

[r-pkgs.had.co.nz/release.html](http://r-pkgs.had.co.nz/release.html)