Sviluppo del Package

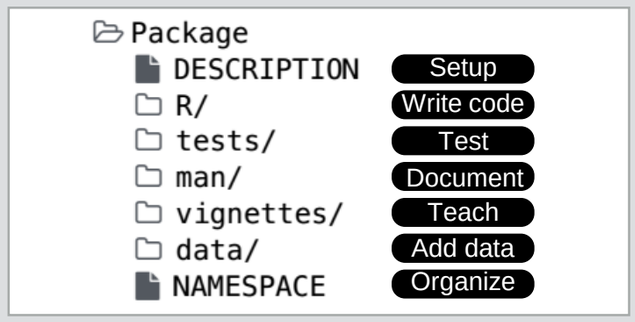
con devtools Scheda di Riferimento

RStudio

Struttura del Package

Un package è una convenzione per organizzare file all'interno delle directory.

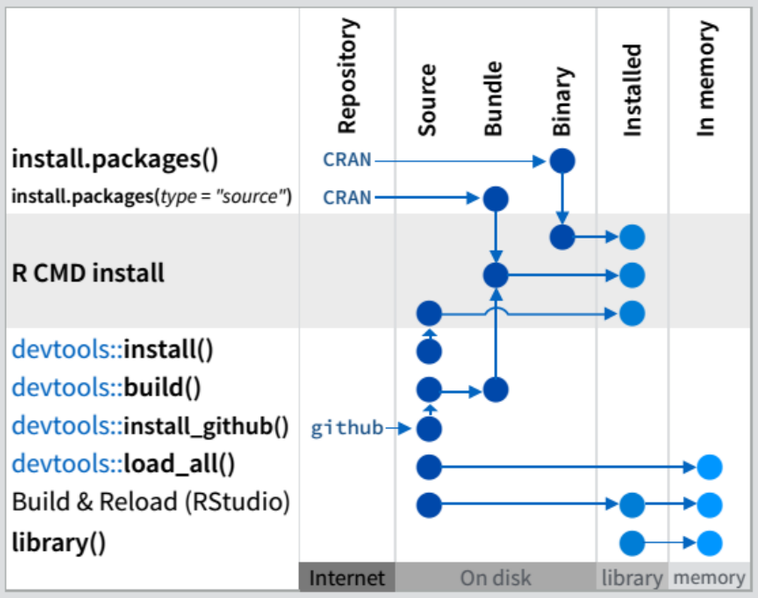
Questa scheda mostra come lavorare con le 7 parti piu comuni di un package R:



I contenuti di un package possono essere salvati su disco come:

- sorgente – un directory con tutto il contenuto (vedi sopra)
- impacchettato – un singolo file compresso (*.tar.gz*)
- binario – un singolo file compresso ottimizzato per uno specifico OS

O installati in una libreria R (caricati in memoria durante una sessione R) o archiviati online in un repository. Le funzioni qui sotto mostrano come spostarsi nei vari stati.



devtools::add_build_ignore(*"file"*)

Aggiunge file a .Rbuildignore, una lista di file che non saranno inclusi nella costruzione del package.

Setup (DESCRIPTION)

Il file DESCRIPTION descrive il lavoro e imposta come il package coopererà con altri packages.

- Devi avere un file DESCRIPTION
- Includi i packages esterni con
  devtools::use_package()
  Aggiunge il package agli Imports field (o Suggests field se il secondo argomento è "Suggests").

CC0	MIT	GPL-2
Nessuna stringa allegata.	Si applica la licenza MIT al codice se condivisa.	Si applica la licenza GPL-2 al codice, *e tutti i codici che lo impacchettano*, se condiviso.

```
Package: mypackage
Title: Title of Package
Version: 0.1.0
Authors@R: person("Hadley", "Wickham", email =
    "hadley@me.com", role = c("aut", "cre"))
Description: What the package does (one paragraph)
Depends: R (>= 3.1.0)
License: GPL-2
LazyData: true
Imports:
    dplyr (>= 0.4.0),
    ggvis (>= 0.2)
Suggests:
    knitr (>= 0.1.0)
```

Importa packages che il tuo package *necessita* per lavorare. R li installerà quando installerà il tuo package.

Suggerisci packages che non sono strettamente necessari al tuo. Gli utenti possono installarli manualmente.

Scrivi Codice (R/)

Tutto il codice R del tuo package va in R/. Un package con solo la directory R/ è comunque utile.

- Crea un nuovo progetto con
  devtools::create(*"path/to/name"*)
  Crea un template per sviluppare nel package.
- Salva il codice in R/ come script (estensione .R)

Workflow

1. Modifica il codice.
2. Carica il codice con
   devtools::load_all()
   Ricarica tutti i file salvati in R/ nella memoria.
   Ctrl/Cmd + Shift + L (keyboard shortcut)
   Salva tutti i file e poi chiama load_all().
3. Prova nella console.
4. Ripeti.

- Usa stile consistente con r-pkgs.had.co.nz/r.html#style
- Clicca su una funzione e premi F2 per aprire la definiz.
- Cerca una funzione con Ctrl + .

Test (tests/)

Usa tests/ per memorizzare i test che ti informaranno se il codice si interrompe.

- Aggiungi una directory tests/ e importa testthat con
  devtools::use_testthat()
  Imposta test automatici con testthat
- Scrivi test con **context(), test()**
- Salva il test come file .R in **tests/testthat/**

Workflow

1. Modifica codice o test.
2. Testa il tuo codice con:
   devtools::test()
   Esegue tutti i test salvati in tests/.
   Ctrl/Cmd + Shift + T
   (keyboard shortcut)
3. Ripeti finchè i test non hanno successo

Example test

```
context("Arithmetic")

test_that("Math works", {
  expect_equal(1 + 1, 2)
  expect_equal(1 + 2, 3)
  expect_equal(1 + 3, 4)
})
```

expect_equal()	È uguale all'interno di una certa tolleranza?
expect_identical()	È esattamente uguale?
expect_match()	Corrispondenza specificata da stringa o expr. regolare?
expect_output()	Stampa specifico output?
expect_message()	Mostra specifico messaggio?
expect_warning()	Mostra specifico avvertimento?
expect_error()	Lancia specifico errore?
expect_is()	L'output eredita da una certa classe?
expect_false()	ritorna FALSE?
expect_true()	ritorna TRUE?

Visita r-pkgs.had.co.nz

Documentazione (man/)

man/ contiene la documentazione delle tue funzioni, le pagine di help nel tuo package.

- Usa i commenti roxygen per commentare ogni funzione accanto alla sua definizione
- Documenta il nome di ogni data set esportato
- Includi esempi utili per ogni funzione

Workflow

1. Aggiungi commenti roxygen nel file .R
2. Converti i commenti roxygen nella documentazione con uno dei seguenti:
   devtools::document()

   Converte i commenti roxygen in file .Rd e li inserice in man/. Costruisce NAMESPACE.

   Ctrl/Cmd + Shift + D (Keyboard Shortcut)
3. Apri help con ? Per l'anteprima della documentazione
4. Ripeti

.Rd formatting tags

```
\emph{italic text}
\strong{bold text}
\code{function(args)}
\pkg{package}

\dontrun{code}
\dontshow{code}
\donttest{code}

\deqn{a + b (block)}
\eqn{a + b (inline)}

\email{name@@foo.com}
\href{url}{display}
\url{url}

\link[=dest]{display}
\linkS4class{class}
\code{\link{function}}
\code{\link[package]{function}}

\tabular{lcr}{
   left \tab centered \tab right \cr
   cell \tab cell     \tab cell
\cr
}
```

Il pacchetto roxygen

roxygen permette di scrivere la documentazione nei file .R con una sintassi abreviata.

- Aggiungi documentazione roxygen come righe di commento che cominciano con #'.
- Piazza il commento direttamente sopra il codice che definisce l'oggetto documentato.
- Piazza un tag roxygen @ tag dopo **#'** per fornire una specifica sezione della documentazione.
- Linee non taggate verranno utilizzare per creare sezione titoli, descrizioni, e dettagli

```
#' Add together two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' @export
add <- function(x, y) {
  x + y
}
```

Tag roxygen comuni

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

Insegna (vignettes/)

vignettes/ contiene documenti che insegnati gli utenti su come risolvere problemi reali con i tuoi tool.

- Crea una directory vignettes/ e un template vignette con **devtools::use_vignette()**

  Aggiungi template vignette come vignettes/my-vignette.Rmd.
- Aggiungi intestazioni YAML alle tue vignettes (vedi a destra)
- Scrivi il bodi delle vignettes in R Markdown (rmarkdown.rstudio.com)

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

Aggiungi Dati (data/)

La directory data/ acconsente di includere dati nel tuo package.

- Conserva i dati in una tra **data/, R/Sysdata.rda, inst/extdata**
- Usa sempre **LazyData: true** nel file DESCRIPTION
- Salva dati come file .Rdata (suggerito)

devtools::use_data()
Aggiunge dati in data/

(R/Sysdata.rda se internal = TRUE)

devtools::use_data_raw()
Aggiunge uno script R usato per pulire il dataset to data-raw/. Include data-raw/ su .Rbuildignore.

Salva dati in

- **data/** per rendere i dati disponibili agli utenti
- **R/sysdata.rda** per mantenere i dati ad uso interno delle funzioni
- **inst/extdata** per rendere disponibili i raw data per il caricamento e il parsing degli esempi. Accedi a questi dati con **system.file()**

Organizza (NAMESPACE)

Il file NAMESPACE permette di rendere il package autosufficiente: non interferirà con altri packages, e altri packages non interferiranno con esso.

- Esporta funzioni inserendo @export nei commenti roxygen
- Importa oggetti da ogni altro packages con package::object (consigliato) o @import, @importFrom, @importClassesFrom, @importMethodsFrom (non sempre consigliati)

Workflow

1. Modifica il tuo codice o test.
2. Documenta tuo package (**devtools::document()**)
3. Controlla NAMESPACE
4. Ripeti finché il NAMESPACE è corretto

Invia il tuo package

r-pkgs.had.co.nz/release.html

Tradotto da Angelo SALATINO • aasalatino@gmail.com • http://salatino.org