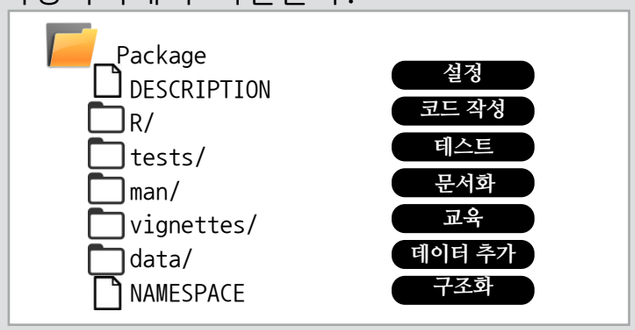
Devtools를 이용한 패키지 개발 커닝쪽지

RStudio

패키지 구조

패키지는 파일을 디렉토리로 구조화하는 관례다.

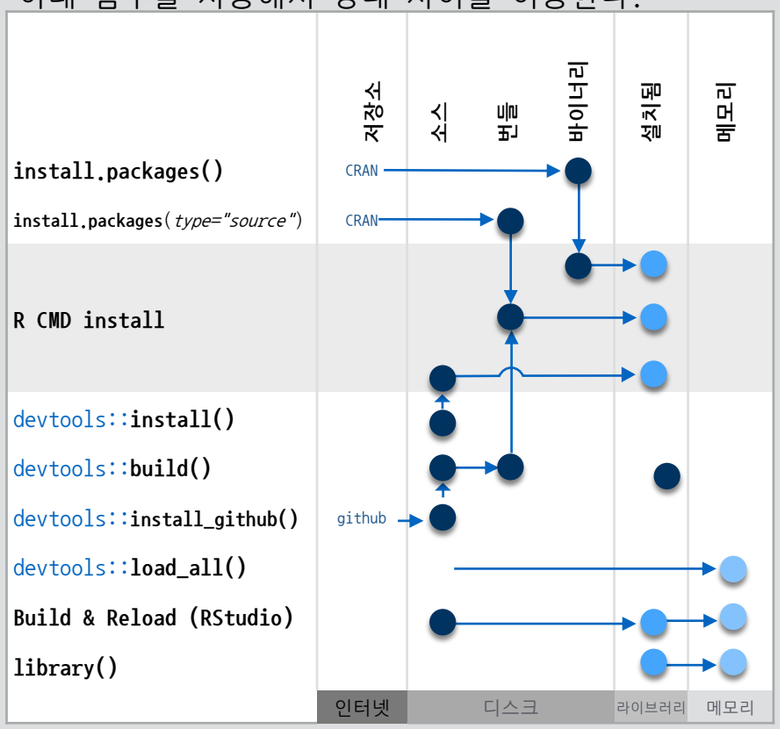
R 팩키지 가장 일반적인 부분을 작업하는 방법을 본 커닝쪽지에서 시연한다.



팩키지 콘텐츠는 디스크에 다음으로 저장될 수 있다:

- **소스** - 하위디렉토리리를 갖는 디렉토리(위처럼)
- **번들** - 단일 앞축 파일 (*.tar.gz*)
- **바이너리** - 특정 OS에 최적화된 단일 압축 파일.

혹은 R 라이브러리 내부에 설치된다 (R 세션 동안 메모리에 적재됨) 혹은 저장소에 온라인으로 보관됨. 아래 함수를 사용해서 상태 사이를 이동한다.



devtools::**add_build_ignore**(*"file"*)
파일을 .Rbuildignore에 추가해서 팩키지 빌드될 때 포함되지 않게함.

설정 (DESCRIPTION)

DESCRIPTION 파일은 작업을 기술하고, 작성한 팩키지가 다른 팩키지와 동작하는 방법을 설정한다.

☑ DESCRIPTION 파일은 필수다.

☑ 다음명령어로 작성한 팩키지가 의존성을 갖는 팩키지를 추가한다.

devtools::**use_package()**
Imports 필드에 팩키지를 추가한다(혹은 만약 두번째 인자가 " Suggests " 면 Suggests 필드)

CC0
어떤 문자열도 부착되지 않음.

MIT
만약 재공유된다면 MIT 라이선스가 작성한 코드에 적용됨.

GPL-2
만약 재공유되다면, GPL-2 라이선스는 작성한 코드에 적용되고 *코드를 번들하는 누구나 모든 코드에* 동일하게 적용된다.

```
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)
```

Import: 작성한 팩키지가 동작하는데 필요한 팩키지를 가져온다. R이 개발한 팩키지를 설치할 때 팩키지를 설치한다.

Suggest: 개발한 팩키지에 그다지 핵심적이지 않은 팩키지를 추천한다. 사용자는 수작업으로 설치할 수 있고, 원하면 설치 않을 수도 있다.

코드 작성 (R/)

작성한 팩키지에 모든 R 코드는 R/ 디렉토리로 들어간다. 단지 R/ 디렉토리만 가진 팩키지도 그 자체로 매우 유용한 팩키지다.

☑ 명령어로 신규 팩키지 프로젝트를 생성한다.

devtools::**create(*"path/to/name"*)**
팩키지 개발위한 템플릿을 생성한다.

☑ 스크립트로 R/ 디렉토리에 작성한 코드를 저장한다 (확장자 .R)

작업흐름

1. 코드를 편집한다.
2. 코드를 다음 둘중 한 명령어로 적재한다.
   devtools::**load_all()**
   R/ 디렉토리에 저장된 모든 파일을 메모리로 적재한다.
   Ctrl/Cmd + Shift + L (키보드 단축키)
   열린 모든 파일을 저장하고나서 load_all() 호출.
3. 콘솔에서 실험한다.
4. 반복한다.

- **r-kgs.had.co.nz/r.html#style** 로 일관된 스타일 사용.
- 함수 정의를 열고자 하면 함수를 클릭하고 **F2**를 누른다.
- **Ctrl +.** 단축키로 함수를 찾는다.

자세한 정보

(英) http://r-pkgs.had.co.nz
(韓)http://r-pkgs.xwmooc.net

테스트 (tests/)

tests/ 디렉토리를 사용해서 만약 코드가 망가지면 정보를 알려줄 수 있는 단위테스트를 저장한다.

☑ **tests/** 디렉토리를 추가하고 **testthat**을 가져온다.
devtools::**use_testthat()**
팩키지 설정을 해서 testthat으로 테스트 자동화한다.

☑ **context()**, **test()**, 예상(expectations)으로 테스트를 작성한다.

☑ .R 파일로 테스트를 tests/testthat/ 디렉토리에 저장한다.

작업흐름

1. 코드 혹은 테스트를 변형.
2. 다음 둘중 하나로 코드를 테스트.
   devtools::**test()**
   tests/ 디렉토리에 *저장된* 모든 테스트를 실행.
   Ctrl/Cmd + Shift + T
   (키보드 단축키)
3. 모든 테스트 통과 때까지 반복.

테스트 예제

```
context("Arithmetic")

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

expect_equal()	적은 허용오차내에서 동일한가?
expect_identical()	정확하게 동일한가?
expect_match()	지정된 문자열 혹은 정규표현식과 매칭되는가?
expect_output()	지정된 출력결과를 출력하는가?
expect_message()	지정된 메시지가 출력되는가?
expect_warning()	지정된 경고가 출력되는가?
expect_error()	지정된 오류를 던지는가?
expect_is()	출력결과가 특정 클래스로부터 상속되었는가?
expect_false()	FALSE를 반환하는가?
expect_true()	TRUE를 반환하는가?

문서 (man/)

man/ 디렉토리에는 개발한 팩키지의 함수와 도움말을 위한 문서가 담겨진다.

- roxygen 주석을 사용해서 정의 옆에 각 함수를 문서화한다.
- 내보내진 각 데이터셋 명칭을 문서화한다.
- 각 함수에 대한 도움말 예제를 포함한다.

작업흐름

1. roxygen 주석을.R 파일에 추가한다.
2. 둘중 한 방법으로 roxygen 주석을 문서로 전환.

   devtools::document()

   roxygen 주석을 .Rd 파일로 전환하고 man/. Builds NAMESPACE 에 위치시킨다.

   Ctrl/Cmd + Shift + D (키보드 단축키)
3. 문서를 미리보려면 ? 로 도움말을 연다.
4. 반복한다.

.Rd 서식 태그

```
\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
}
```

roxygen 팩키지

roxygen을 사용해서 약칭 구문으로 .R 파일에 인라인 방식으로 문서를 작성한다.

- #' 으로 시작되는 주석 라인으로 roxygen 문서를 추가한다.
- 문서화될 객체를 정의하는 코드 위에 직접 주석라인을 위치시킨다.
- #' 뒤에 @ 태그(우측)를 위치해서 문서 특정부분 정보를 제공한다.
- 태그되지 않은 라인은 제목, 기술, 상세한 부분정보를 생성하는데 사용된다 (순서대로).

```
#' 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
}
```

자주쓰는 roxygen 태그

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

교육 (vignettes/)

vignettes/ 디렉토리에 문서가 담겨져서 사용자가 개발된 도구로 실제 문제를 해결하는 방법을 가르친다.

- vignettes/ 디렉토리와 템플릿 소품문을 생성한다.

  devtools::use_vignette()

  vignettes/my-vignette.Rmd으로 템플릿 소품문을 추가
- (우측처럼) YAML헤더를 작성한 소품문에 추가한다.
- R Markdown으로 소품문 본문을 작성한다. (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}
---
```

데이터 추가 (data/)

data/ 디렉토리는 개발 팩키지에 데이터를 포함하는데 사용한다.

- 데이터를 **data/, R/Sysdata.rda, inst/extdata** 중 한곳에 저장한다.
- DESCRIPTION 파일에 항상 **LazyData: true**로 설정한다.
- 데이터를 .Rdata 파일로 저장한다 (추천)

devtools::use_data()

데이터객체를 data/ 디렉토리에 추가한다. (R/Sysdata.rda if **internal = TRUE**)

devtools::use_data_raw()

data-raw/ 디렉토리에 데이터셋을 정제하는데 사용한 R 스크립트를 추가한다. .Rbuildignore. 파일에 data-raw/ 디렉토리를 추가한다.

데이터를 다음에 저장한다.

- **Data/**: 팩키지 사용자에게 데이터를 이용하게함.
- **R/sysdata.rda**: 작성한 함수가 내부적으로 사용하게 데이터 저장
- **inst/extdata**: 예제를 적재, 파싱하는데 원데이터를 이용가능하게 함. system.file()로 데이터에 접근

구조화 (NAMESPACE)

NAMESPACE 파일은 개발된 팩키지가 독립적으로 동작하게 도움: 다른 팩키지를 간섭하지 못하게 하고, 다른 팩키지가 간섭하지 못하게 한다.

- roxygen 주석에 @export 태그를 두어 사용자를 위한 함수를 내보내게 한다.
- **package::object** (추천) 혹은 **@import, @importFrom, @importClassesFrom, @importMethodsFrom** (항상 추천하지 않음)를 사용해서 다른 팩키지에서 객체를 가져온다.

작업흐름

1. 코드 혹은 테스트를 변형한다.
2. 개발 팩키지를 문서화(**devtools::document()**)
3. NAMESPACE를 검사한다.
4. NAMESPACE가 올바를 때까지 반복한다.

팩키지 제출

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