Packages are easy!

Hadley Wickham

@hadleywickham

Chief Scientist, RStudio

May 2014

A package is a set of conventions that (with the right tools) makes your life easier

A package is a set of conventions that (with the right tools) makes your life easier

```
# The right tools:
#
# The latest version of R.
# RStudio.
#
# Code development tools:
#
# * Windows: Rtools, download installer from
#   http://cran.r-project.org/bin/windows/Rtools/
# * OS X: xcode, free from the app store
# * Linux: apt-get install r-base-dev (or similar)
#
# Packages that make your life easier:
install.packages(c("devtools", "knitr", "Rcpp",
  "roxygen2", "testthat"))
```

A package is a set of conventions that (with the right tools) makes your life easier

R/

R code

```
devtools::create("easy")
# Creates DESCRIPTION
# Creates rstudio project
# Sets up R/ directory

# NB: All devtools functions take a path to a
# package as the first argument. If not supplied,
# uses the current directory.
```

Never use `package.skeleton()`!

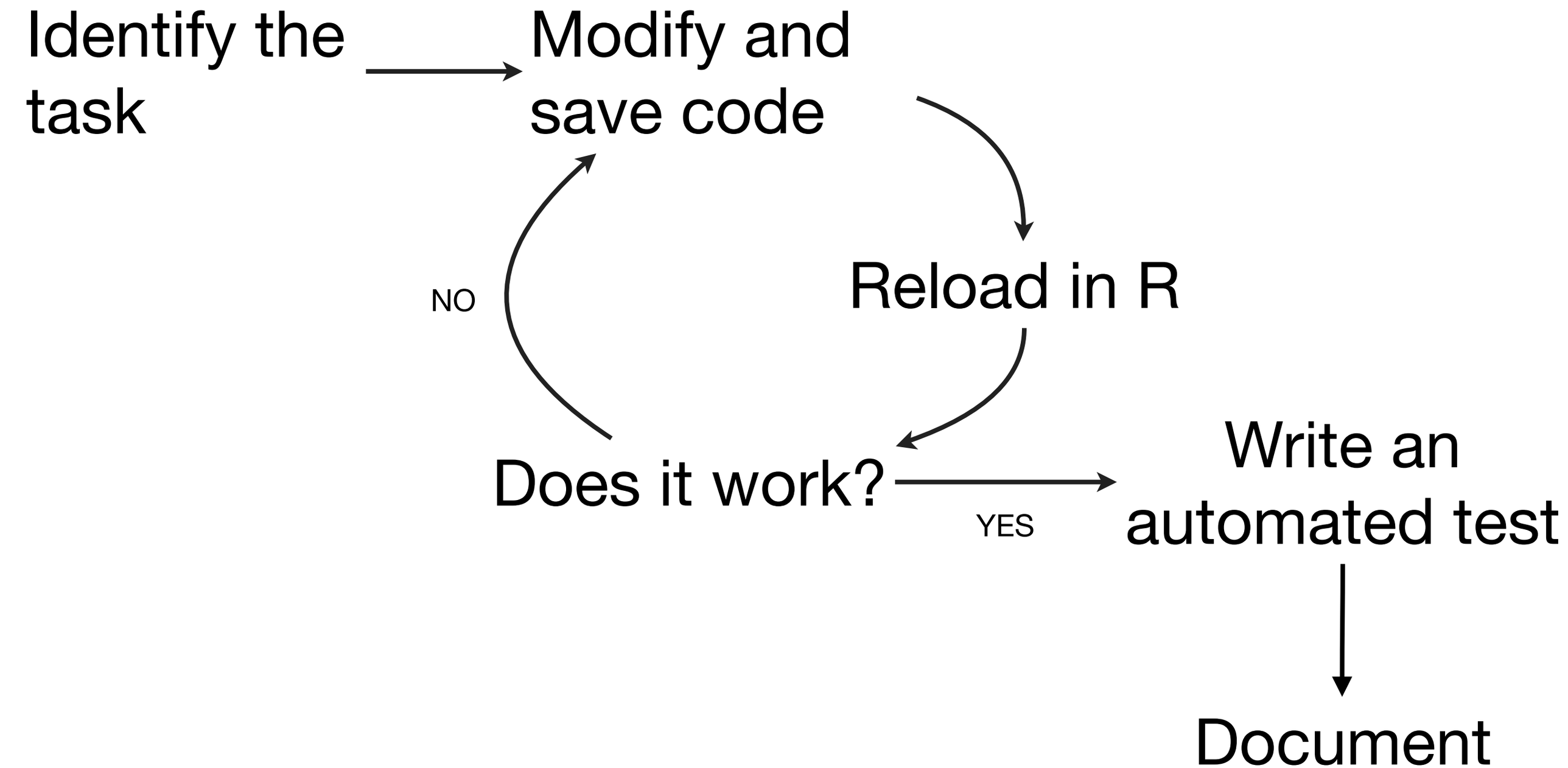
Why? It only works once, it does something v. easy to do by hand, and automates a job that needs to be manual

```
# Add some R code

load_all()

# In rstudio: cmd + shift + l
# Also automatically saves all files for you
```

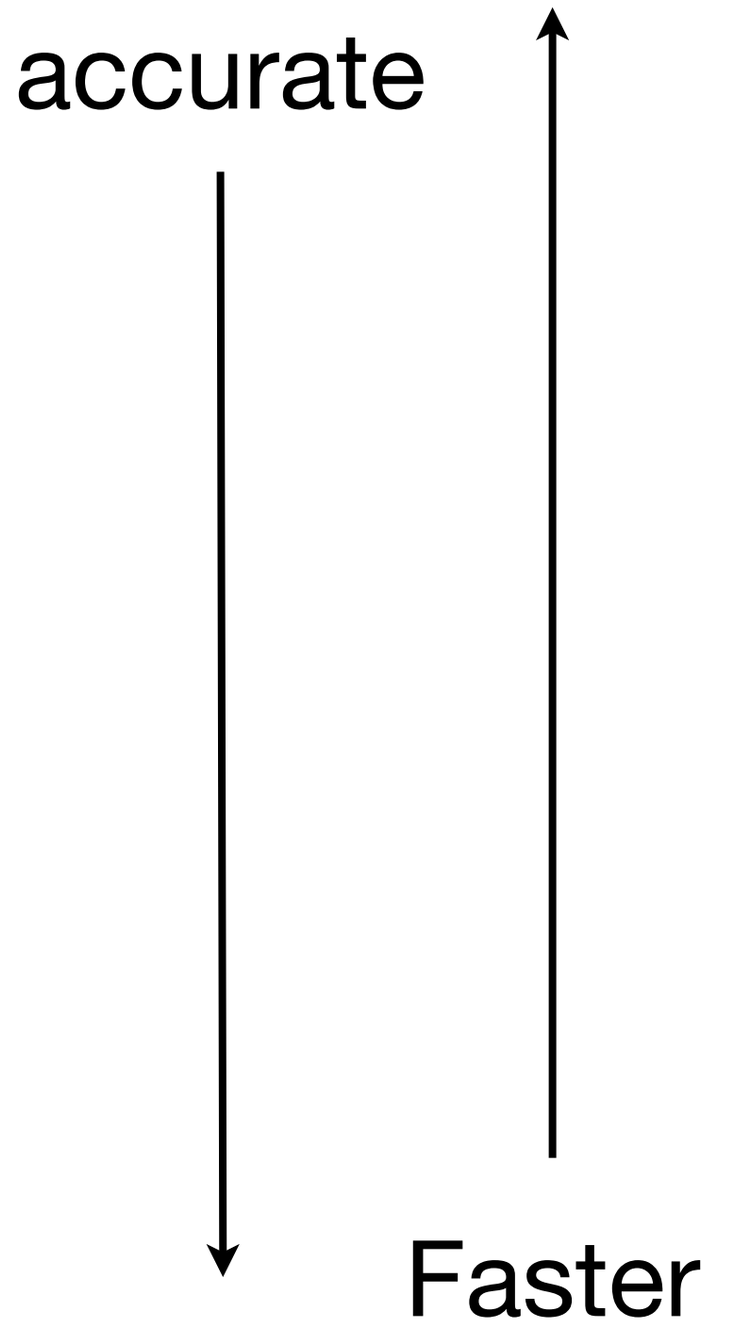
Programming cycle



```
# Reloads changed code, data, …
# cmd + shift + l
load_all()

# Rstudio: Build & Reload
# cmd + shift + b
# Installs, restarts R, reloads

# Builds & installs
install()
# (needed for vignettes)
```



DESCRIPTION

Who can use it, what it needs, and who wrote it

```
Package: easy
Title: What the package does (short line)
Version: 0.1
Authors@R: "First Last <first.last@example.com> [aut, cre]"
Description: What the package does (paragraph)
Depends: R (>= 3.1.0)
License: What license is it under?
LazyData: true
```

We're done!

That's all you *need* to know about packages

But you can also add data, *documentation*, unit tests, *vignettes* and C++ code

+

man/

Compiled documentation

Roxygen2

- Essential for function level documentation. Huge time saver
- R comments → Rd files → human readable documentation
- `Rd2roxygen` package converts Rd to roxygen if you have legacy packages

Raw R source

```
#' Order a data frame by its columns.
#'
#' This function completes the subsetting, transforming and ordering triad
#' with a function that works in a similar way to \code{\link{subset}} and
#' \code{\link{transform}} but for reordering a data frame by its columns.
#' This saves a lot of typing!
#'
#' @param df data frame to reorder
#' @param ... expressions evaluated in the context of \code{df} and
#'   then fed to \code{\link{order}}
#' @keywords manip
#' @export
#' @examples
#' mtcars[with(mtcars, order(cyl, disp)), ]
#' arrange(mtcars, cyl, disp)
#' arrange(mtcars, cyl, desc(disp))
arrange <- function(df, ...) {
  ord <- eval(substitute(order(...)), df, parent.frame())
  unrowname(df[ord, ])
}
```

Raw R source

```
#' Order a data frame by its columns.
#'
#' This function completes the subsetting, transforming and ordering triad
#' with a function that works in a similar way to \code{\link{subset}} and
#' \code{\link{transform}} but for reordering a data frame by its columns.
#' This saves a lot of typing!
#'
#' @param df data frame to reorder
#' @param ... expressions evaluated in the context of \code{df} and
#'   then fed to \code{\link{order}}
#' @keywords manip
#' @export
#' @examples
#' mtcars[with(mtcars, order(cyl, disp)), ]
#' arrange(mtcars, cyl, disp)
#' arrange(mtcars, cyl, desc(disp))
arrange <- function(df, ...) {
  ord <- eval(substitute(order(...)), df, parent.frame())
  unrowname(df[ord, ])
}
```

Generated Rd file

```
\name{arrange}
\alias{arrange}
\title{Order a data frame by its columns.}
\usage{arrange(df, ...)}

\description{
  Order a data frame by its columns.
}

\details{
  This function completes the subsetting, transforming and
  ordering triad with a function that works in a similar
  way to \code{\link{subset}} and \code{\link{transform}}
  but for reordering a data frame by its columns. This
  saves a lot of typing!
}
\keyword{manip}
\arguments{
  \item{df}{data frame to reorder}
  \item{...}{expressions evaluated in the context of \code{df} and then fed
to \code{\link{order}}}
}
\examples{mtcars[with(mtcars, order(cyl, disp)), ]
arrange(mtcars, cyl, disp)
arrange(mtcars, cyl, desc(disp))}
```

Html view in R

R: Order a data frame by its

127.0.0.1:12111/library/plyr/html/arrange.html

arrange {plyr} R Documentation

Order a data frame by its colums.

Description

This function completes the subsetting, transforming and ordering triad with a function that works in a similar way to `subset` and `transform` but for reordering a data frame by its columns. This saves a lot of typing!

Usage

```
arrange(df, ...)
```

Arguments

`df`	data frame to reorder
`...`	expressions evaluated in the context of `df` and then fed to `order`

Examples

```
mtcars[with(mtcars, order(cyl, disp)), ]
arrange(mtcars, cyl, disp)
arrange(mtcars, cyl, desc(disp))
```

[Package *plyr* version 1.6 Index]

Documentation cycle

1. Update roxygen comments.
2. `document()` (Cmd + Shift + D)
3. `?topic`

+

vignettes/

Long-form documentation

markdown + knitr

Easy to write

Doesn’t need latex toolchain

Only available ≥3.0.0

By Yihui Xie

```
use_knitr()

# Adds to DESCRIPTION
VignetteBuilder: knitr
Suggests: knitr
```

```
# In each .Rmd file in vignettes/
<!--
%\VignetteEngine{knitr}
%\VignetteIndexEntry{Vignette title}
-->
```

````
<!--
%\VignetteEngine{knitr}
%\VignetteIndexEntry{Vignette title}
-->

# Introduction to my package

The easy package provides a number of simple functions that make it easy to
access small integers.

For example the `one()` function returns the number one:

```{r}
library(easy2)
one()
```

Or in a plot:

```{r}
plot(one())
```
````

```
# Preview with cmd + shift + k (knitr)
# Vignette runs in separate (clean) session,
# so make sure to build & reload first.

# To check actually built in package, use
# install().
install()
browseVignettes(package = "easy")
```

Introduction to my package

file://localhost/Users/hadley/Documents/devtools/13-easy-...

Introduction to my package

The easy package provides a number of simple functions that make it easy to access small integers.

For example the one() function returns the number one:

```
library(easy2)
one()
```

```
## [1] 1
```

Or in a plot:

```
plot(one())
```

+

src/

C++ code

Rcpp

- Don't have time to talk about it today, but if you're interesting in writing fast code, learn Rcpp!
  - Get set up with `use_rcpp()`
  - My guide at http://adv-r.had.co.nz/Rcpp.html
  - By Dirk Eddelbuettel, Romain Francois, and others

+

inst/tests/

Unit tests

testthat

- Likewise, no time to talk about unit testing, but it's a great way to prevent bugs.
- You are already doing testing, learn how to make it formal.
- Get setup with `use_testthat()`
- My guide at adv-r.had.co.nz/Testing.html

Test cycle

1. Update code.
2. Update tests.
3. `test()` (Cmd + Shift + t)

+

NAMESPACE

what functions your package needs, and
what functions it provides

Motivation

- What happens if two packages both have a function with the same name?
- Namespaces provide a way to resolve this issue, and to reduce it.
- Splits functions into internal and external

```
library(plyr)
library(Hmisc)
is.discrete

library(Hmisc)
library(plyr)
is.discrete

Hmisc::is.discrete
plyr::is.discrete
```

Note

- Mostly important for CRAN: be a good neighbour to other packages.
- You need to know about it, but it's a bit tricky.
- Roxygen2 makes it relatively easy with @export and @importFrom

```
# To use a function from another package:

# Add package name (and version) to DESCRIPTION
Imports: ggplot2 (>= 0.9.3)

# Add @importFrom directive next to function
#' @importFrom ggplot2 ggplot

# Or import all functions from a package with
#' @import ggplot2
```

```
# To export a function from your package use
#' @export
# By default load_all() makes all functions
# available, even if not exported. Use
# load_all(export_all = FALSE) to only reveal
# exported functions. This is useful for testing.
```

Never use depends!

Why? Because it exposes many extra functions to the user, massively increasing the chances of clashes

More to learn

Learn from others

If you only remember one thing:

Read the source code of other packages.

These are the packages I'm most proud of:

`https://github.com/hadley/plyr`

`https://github.com/hadley/stringr`

`https://github.com/hadley/devtools`

`https://github.com/hadley/lubridate`

`https://github.com/hadley/evaluate`

`https://github.com/hadley/reshape`

Distribution

- Easiest way: put on github and use `devtools::install_github()` to install
- Most rigorous (and painful): put on CRAN. See `check()` and `release()` for more details

devtools

- devtools is constantly improving as I figure out where the pain points are
- Use `install_github("devtools")` to get the latest version
- If something doesn't work for you, please file a bug at `github.com/hadley/devtools/issues`