Documentation & namespaces

Hadley Wickham

Assistant Professor / Dobelman Family Junior Chair

Department of Statistics / Rice University

October 2011

1. Package documentation
2. Function documentation
3. Text formatting
4. Exports
5. Imports

Package level

Package-level

README

NEWS

Package help topic

Vignettes + `CITATION`

Demos

https://github.com/hadley/devtools/wiki/docs-package

README

Should outline the basics of your package: what does it do, and why should people care?

You'll also use this when you email out package release announcements.

Particularly important if your package is on github.

`plyr, lubridate, stringr`

NEWS

Whenever you make a change to your package, make sure to make a note of it in the `NEWS` file.

R has a special format for `NEWS`, but it's poorly documented. Use `show_news()` to check it.

Use in release announcements.

`plyr`, `stringr`, `ggplot2`

ggplot2 0.9.0 Version number

MINOR CHANGES Subheadings (optional)

* `geom_text` now supports `fontfamily`, `fontface`, and `lineheight`
  aesthetics for finer control over text display. (Thanks to Kohske Takahashi
  for the patch. Fixes #60)

* `collide`, which powers `position_dodge` and `position_stack`, now does not
  error on single x values (Thanks to Brian Diggs for a fix. #157)

* When printing a ggplot2 object, the rendered plot information is returned
  invisibly. You can capture this with (e.g.) `x <- print(qplot(mpg, wt, data
  = mtcars))` and in the future will be able to use it to get information
  about the plot computations, such as the range of all the scales, and the
  exact data that is plotted.

Individual items indented and start with *

* `scale_shape` finally returns an error when you try and use it with a
  continuous variable

DEVELOPMENT

* ggplot2 has moved away from the two (!!) homegrown documentation systems
  that it previously relied on, and now uses roxygen extensively. The current
  downside is that this means that ggplot2 website can no longer be updated,
  but I hope work with the `helpr` package will resolve that shortly.

Package help topic

You should be able to get help on the package from within R using `?mypackage` or `package?mypackage`.

This is created in the same way as function level documentation so we'll discuss in the next section.

`lubridate`

Vignettes

A vignette is a long-form document that introduces new users to your package.

If you write a nice one, it's worth publishing in JSS or the R-journal.

When published, use a `CITATION` file so users know what to cite.

I am bad at writing vignettes.

Your turn

Inspect the `coin` package. How many vignettes are there? How are they different from ordinary latex files?

Vignettes

Must be written in Sweave

http://cran.r-project.org/doc/manuals/R-exts.html#Writing-package-vignettes

http://www.statistik.lmu.de/~leisch/Sweave/Sweave-Rnews-2003-2.pdf

Demos

Long-form R scripts that show how all the pieces of your package fit together.

Function level

Roxygen2

- Essential for function level documentation. Huge time saver
- Converts source comments into the Rd files needed for R packages
- `Rd2roxygen` package converts Rd to roxygen if you have legacy packages

`https://github.com/hadley/devtools/wiki/docs-function`

```
#' 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, ])
}
```

```
#' 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, ])
}
```

```
\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))}
```

.rd_roclet

Tag	Purpose
`@param`	Describe inputs
`@examples`	Show how the function works
`@author`	Who wrote the function (if different from package)
`@seealso`	Pointers to related functions
`@return`	Describe outputs
`@aliases`	Make it easier for users to find
`@rdname`	Useful for functions that are invalid filenames and for combining docs

Documentation cycle

1. Update roxygen comments.
2. `document()`
3. `check_doc()`
4. `install()`
5. Inspect documentation.

```
document("hof-doc")
check_doc("hof-doc")
install("hof-doc") # sometimes requires R restart
?tee
```

Your turn

Familiarise yourself with the roxygen documentation for `hof-doc`.

Make a few small changes and then run `document()`. How do the Rd files change?

Install the package (with `install()`) and read the documentation. What works well? What could I have done better?

Your turn

Add a package help topic to `hof-doc`, following the description in `https://github.com/hadley/devtools/wiki/docs-package` and using other packages as needed for reference.

Re-run `document()`, install and check.

Your turn

In `hof-doc`, document the Ask function.

```
document("hof-doc")
# Look at Ask.Rd
install("hof-doc")
?Ask
```

```
Ask <- function(f, x, ...) {
  ind <- vapply(x, f, logical(1), ...)
  na.false(ind)
}
```

```
na.false <- function(x) {
  !is.na(x) & x
}
```

```r
elements <- list(1:10, c(-1, 10), c(TRUE, FALSE),
  letters)
results <- lapply(elements, log)
results <- lapply(elements, function(x) try(log(x)))

is.error <- function(x) inherits(x, "try-error")
```

```r
# Just to get the successful ones
successful <- Filter(Negate(is.error), results)

# Which ones failed?
failures <- vapply(results, is.error, logical(1))
failures <- Ask(is.error, results)
```

Text formatting

Tag	Purpose
`\code{}`	Discus R code
`\link{}`	Make link to another function. Usually wrapped in `\code{}`

`\enumerate{`
`  \item First item`
`}`
	Numbered list. Use `\itemize{}` for bulleted.

`\eqn{}`	Inline equation (standard latex)
`\emph{}`	Italic text
`\strong{}`	Bold text

More tips

In examples, `\dontrun{}` allows you to choose not to automatically run parts of an example. This is useful if they cause errors.

Exports

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

https://github.com/hadley/devtools/wiki/Namespaces

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

library(Hmisc)
library(plyr)
is.discrete

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

Conflicts

Two problems:

User needs to explicitly specify which function to use. Can’t solve automatically.

Developer worries that their package will use the wrong function. Can solve automatically!

Defaults

If you don’t do anything, all functions in your package will be made available to your users.

You probably don’t want to do this as packages with fewer functions are easier to use, and any **external** function is harder to change in the future.

Internal	External
Only for use within package	For use by others
Documentation optional	Must be documented
Easily changed	Changing will break other peoples code

Exporting

- `@export` adds a function to the namespace
- S3 methods exported as long as you use have `@export` and `@method methodname classname`
- S4 currently not well supported. (Hopefully soon!)

`load_all()`

- Ignores namespaces
- Probably what you want for development, but can hide namespace problems until you run `R CMD check`
- Be aware!

Imports

Dependencies

- What do you do if your package needs another package to work?
- There are three ways of recording this fact in your `DESCRIPTION`

Dependencies

`Depends`: this package is required for your package to work, and will be loaded when your package is loaded

`Imports`: also required, but package won't be loaded. Best practice. Must also been included in `NAMESPACE`

`Suggests`: used in only a few places. Must also be loaded when needed with `require()`

```
# Currently
library(ggplot2)

?cast

# When you load ggplot2 all functions in the
# packages that ggplot2 depends on are also loaded.
# Increases the chance of a conflict!

# Generally, better to be explicit than implicit
library(ggplot2)
library(plyr)
library(reshape2)
```

Importing

- Add package to `Imports` in `DESCRIPTION`
- Add `@imports package` to your package documentation
- Minimises chances your package will not interfere with others – good development practice