**What is a vignette? Where does it live?**

In this section we shall go over basics of package vignettes.

In practice, if your package contains one or several vignette(s), an user could

* find them using the vignette() or browseVignettes() function, for instance they could type vignette(package = "rhub") or browseVignettes(package = "rhub") to access the list of installed vignettes for the rhub package[2](https://blog.r-hub.io/2020/06/03/vignettes/#fn:2)

vignette(package = "rhub")

| **Item** | **Title** |
| --- | --- |
| rhub | get-started (source, html) |
| local-debugging | Local Linux checks with Docker (source, html) |

browseVignettes("rhub")

| **Vignette** | **Title** |
| --- | --- |
| rhub.html | get-started |
| local-debugging.html | Local Linux checks with Docker |

As a package author you could be fine only knowing about [usethis::use\_vignette()](https://usethis.r-lib.org/reference/use_vignette.html) for creating a vignette, and that packages used in the vignette need to be listed in DESCRIPTION (under Suggests if they’re only used in the vignette[3](https://blog.r-hub.io/2020/06/03/vignettes/#fn:3)). Still, it’s useful to know about vignettes for debugging problems or finding workarounds for issues you might encounter.

**Infrastructure & dependencies for vignettes**

The building of package vignettes can either use the default Sweave vignette engine, or [a vignette engine provided by a CRAN package](https://cran.r-project.org/doc/manuals/r-release/R-exts.html#Non_002dSweave-vignettes)

[See the source of rhub main vignette](https://github.com/r-hub/rhub/blob/master/vignettes/rhub.Rmd). It has YAML metadata at the top, some non-executed code chunks, some executed code chunks. To allow for that vignette to be built, a [field in DESCRIPTION](https://github.com/r-hub/rhub/blob/6ae6f35e958f3beab1e2c8e6f704affa23c8ce29/DESCRIPTION#L47) mentions the vignette engine:

VignetteBuilder: knitr, rmarkdown

And these two packages are declared as dependencies under Suggests as well.

The creation of a boilerplate Rmd under a new vignettes folder, and the dependencies declaration in DESCRIPTION, are what usethis::use\_vignette() would handle for you. Then you can write as you would a standard R Markdown document, knitting for previewing it.

Other vignette builders include R.rsp that we’ll mention again later, noweb to use the noweb literate programming tool (which actually looks a lot like sweave?), rasciidocs that was recently archived at the time of writing. It is unlikely you’ll want to write your own vignette engine.

How many packages use a non-Sweave vignette? One way to assess that is to look for packages that have a VignetteBuilder field in DESCRIPTION with R-hub’s own pkgsearch.

results <- pkgsearch::advanced\_search("\_exists\_" = "VignetteBuilder")

attr(results, "metadata")$total

[1] 4969

knitr <- pkgsearch::advanced\_search(VignetteBuilder = "knitr")

attr(knitr, "metadata")$total

[1] 4739

# for comparison

nrow(available.packages())

[1] 15694

Quite a lot, about 32% of CRAN pages use a non Sweave vignette engine and about 30% use knitr for at least one vignette[6](https://blog.r-hub.io/2020/06/03/vignettes/#fn:6) Other packages might have *Sweave* vignettes, and some CRAN packages don’t have vignettes, whereas having a vignette is compulsory for Bioconductor packages.

**Overview of vignettes states**

Following the neat diagram of the R packages book,

* You write your vignette(s) in the vignettes/ folder.
* During building vignettes are built and then vignettes sources, outputs, and anything written in install\_extras (a friend of .Rbuildignore and .Rinstignore except it shows what to *keep* not *discard*!) gets moved to inst/doc/.

If your vignette shows an external image not generated by the build process, you also need to include it in install\_extras,

* During installation the content of inst/doc/ get copied to doc/. (See e.g. rhub content in my local library:)

fs::dir\_tree(find.package("rhub"))

/home/maelle/R/x86\_64-pc-linux-gnu-library/3.6/rhub

├── DESCRIPTION

├── INDEX

├── LICENSE

├── Meta

│ ├── Rd.rds

│ ├── features.rds

│ ├── hsearch.rds

│ ├── links.rds

│ ├── nsInfo.rds

│ ├── package.rds

│ └── vignette.rds

├── NAMESPACE

├── NEWS.md

├── R

│ ├── rhub

│ ├── rhub.rdb

│ └── rhub.rdx

├── bin

│ ├── rhub-linux-docker.sh

│ └── rhub-linux.sh

├── doc

│ ├── index.html

│ ├── local-debugging.R

│ ├── local-debugging.Rmd

│ ├── local-debugging.html

│ ├── rhub.R

│ ├── rhub.Rmd

│ └── rhub.html

├── help

│ ├── AnIndex

│ ├── aliases.rds

│ ├── figures

│ │ └── logo.png

│ ├── paths.rds

│ ├── rhub.rdb

│ └── rhub.rdx

└── html

├── 00Index.html

└── R.css

**Your vignette for R CMD check**

So, sometimes R CMD check[7](https://blog.r-hub.io/2020/06/03/vignettes/#fn:7) will throw errors related to vignette building. How to deal with them?

Based on what we said in the previous subsection, R CMD build builds vignettes from vignettes/ whereas R CMD check checks they can be rebuilt from inst/doc/. So if there were data in vignettes/, given it’s not copied to inst/doc/… R CMD check will error!

It’s also useful to know that there are options related to vignette building and checking in R CMD build and R CMD check. Of course you don’t control these options for CRAN but you do control them when sending your packages to R-hub package builder, and when setting up continuous integration.

R CMD build --no-build-vignettes --no-manual .

R CMD check --no-manual --ignore-vignettes --as-cran \*. tar.gz

For R-hub package builder,

* To tweak the build you need to build your package yourself (from the command line or with devtools::build()) and indicate the path to the tarball, as opposed to the package source, in your call to rhub::check()
* You can tweak the R CMD check by using the check\_args argument.

**Workaround workflows for vignettes**

In this section we’ll go over workarounds for some common vignette “problems”.

**How to include my pre-print / cheatsheet as a PDF vignette?**

Sometimes you might want to include a PDF as a vignette, without wanting to deal with missing LaTeX dependencies; or because the PDF is not knit from R (a cheatsheet); or the computations are too long. In that case there are two alternatives:

VignetteBuilder:

knitr,

R.rsp

%\VignetteIndexEntry{Heck, Arnold, & Arnold (2018): TreeBUGS paper (Behavior Research Methods)}

%\VignetteEngine{R.rsp::asis}

%\VignetteKeyword{PDF}

%\VignetteKeyword{HTML}

%\VignetteKeyword{vignette}

%\VignetteKeyword{package}

%\VignetteKeyword{TreeBUGS}

**How to include a compute-intensive / authentication-dependent vignette?**

A very similar problem can happen with HTML vignettes, when their computations are too long, or depend on a system dependency or authentication token absent from CRAN machines – hence R CMD check would fail for sure. So, what can you do?

```{r, echo = FALSE}

NOT\_CRAN <- identical(tolower(Sys.getenv("NOT\_CRAN")), "true")

knitr::opts\_chunk$set(

collapse = TRUE,

comment = "#>",

purl = NOT\_CRAN,

eval = NOT\_CRAN

)

```

**Hey what about testing? And reproducibility?**

Some may argue that your package is not fully tested this way, but that depends on how well your package tests/ are written. I tend to look at examples() and vignettes as demos, and tests/ as actually tests. All should of course pass R CMD check and run, but the tests/ are what really test the package.

He also makes the point,

For reproducibility, I would include the root/source vignette in the package as well, e.g. in inst/full-vignettes/ with instructions and/or a function on how to rebuild it.

**User-friendly vignettes**

In this section we’ll give some tips for making vignettes easier to navigate.

**Pretty vignettes**

You might want to tweak layout and aspect of your vignette a bit to make people even more likely to read them, maybe with custom CSS. Using a disappointingly unspecific GitHub code search on R-hub mirror of CRAN we found the example of idiogramFISH that defines and uses custom stylesheets for its vignette, that makes the vignette look very modern on its CRAN page! Note that it also uses some JavaScript for the table of content and “return to top” links, definitely not light-weight styling.

Now, an even better way to tweak your vignettes is to invest some time in creating a pkgdown website that will feature both manual pages, vignettes, changelogs, etc. It’s actually little work. It’s worth it reading how vignettes are built in pkgdown docs, in particular

* A vignette called packagename.Rmd will appear under “Get started” in the navbar;
* You can tweak the navbar.

Some further thoughts around vignettes and pkgdown. Since vignettes look better and are more integrated with other docs in the pkgdown website than locally, should your local vignettes contain a link to the pkgdown version to be sure that users that look at an offline vignette but have an internet connection can get a better user experience? And regarding the offline experience, would it make sense to also generate a PDF version of HTML vignettes, maybe with paged.js?