

Package ‘pkgdown’

December 2, 2021

Title Make Static HTML Documentation for a Package

Version 2.0.1

Description Generate an attractive and useful website from a source package. 'pkgdown' converts your documentation, vignettes, 'README', and more to 'HTML' making it easy to share information about your package online.

License MIT + file LICENSE

URL <https://pkgdown.r-lib.org>, <https://github.com/r-lib/pkgdown>

BugReports <https://github.com/r-lib/pkgdown/issues>

Depends R (>= 3.1.0)

Imports bslib,
callr (>= 2.0.2),
crayon,
desc,
digest,
downlit (>= 0.4.0),
fs (>= 1.4.0),
httr (>= 1.4.2),
jsonlite,
magrittr,
memoise,
purrr,
ragg,
rlang (>= 0.4.0),
rmarkdown (>= 1.1.9007),
tibble,
whisker,
withr,
xml2 (>= 1.3.1),
yaml

Suggests covr,
diffviewer,
evaluate,
htmltools,
htmlwidgets,
knitr,
lifecycle,

```

methods,
openssl,
pkgload (>= 1.0.2),
rsconnect,
rstudioapi,
rticles,
sass,
testthat (>= 3.0.0),
tools

```

VignetteBuilder knitr

Config/testthat/edition 3

Config/potools/style explicit

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

SystemRequirements pandoc

R topics documented:

as_pkgdown	2
build_articles	3
build_favicons	6
build_home_index	7
build_news	11
build_reference	13
build_search	16
build_site	16
build_site_github_pages	24
build_tutorials	25
clean_site	26
deploy_to_branch	26
init_site	27
in_pkgdown	27
pkgdown_sitrep	28
preview_site	28
rd2html	29
render_page	29
template_navbar	30

as_pkgdown

Generate pkgdown data structure

Description

You will generally not need to use this unless you need a custom site design and you're writing your own equivalent of `build_site()`.

Usage

```
as_pkgdown(pkg = ".", override = list())
```

Arguments

pkg	Path to package.
override	An optional named list used to temporarily override values in <code>_pkgdown.yml</code>

build_articles	<i>Build articles section</i>
----------------	-------------------------------

Description

`build_articles()` renders each R Markdown file underneath vignettes/ and saves it to articles/. There are two exceptions:

- Files that start with `_` (e.g., `_index.Rmd`) are ignored, enabling the use of child documents in [bookdown](#)
- Files in vignettes/tutorials are handled by [build_tutorials\(\)](#)

Vignettes are rendered using a special document format that reconciles `rmarkdown::html_document()` with the pkgdown template. This means articles behave slightly differently to vignettes, particularly with respect to external files, and custom output formats. See below for more details.

Note that when you run `build_articles()` directly (outside of `build_site()`) vignettes will use the currently installed version of the package, not the current source version. This makes iteration quicker when you are primarily working on the text of an article.

Usage

```
build_articles(
  pkg = ".",
  quiet = TRUE,
  lazy = TRUE,
  override = list(),
  preview = NA
)
```

```
build_article(name, pkg = ".", data = list(), lazy = FALSE, quiet = TRUE)
```

```
build_articles_index(pkg = ".")
```

Arguments

pkg	Path to package.
quiet	Set to FALSE to display output of knitr and pandoc. This is useful when debugging.
lazy	If TRUE, will only re-build article if input file has been modified more recently than the output file.
override	An optional named list used to temporarily override values in <code>_pkgdown.yml</code>

preview	If TRUE, or <code>is.na(preview) && interactive()</code> , will preview freshly generated section in browser.
name	Name of article to render. This should be either a path relative to vignettes/ without extension, or index or README.
data	Additional data to pass on to template.

Index and navbar

You can control the articles index and navbar with a `articles` section in your `_pkgdown.yaml`. It defines a list of sections, each of which can contain four fields:

- `title` (required): title of section, which appears as a heading on the articles index.
- `desc` (optional): An optional markdown description displayed underneath the section title.
- `navbar` (optional): A couple of words used to label this section in the navbar. If omitted, this section of vignettes will not appear in the navbar.
- `contents` (required): a list of article names to include in the section. This can either be names of individual vignettes or a call to `starts_with()`. The name of a vignette includes its path under vignettes without extension so that the name of the vignette found at `vignettes/pizza/slice.Rmd` is `pizza/slice`.

The title and description of individual vignettes displayed on the index comes from `title` and `description` fields of the YAML header in the Rmds.

For example, this yaml might be used for some version of dplyr:

```
articles:
- title: Main verbs
  navbar: ~
  contents:
  - one-table
  - two-table
  - rowwise
  - colwise

- title: Developer
  desc: Vignettes aimed at package developers
  contents:
  - programming
  - packages
```

Note the use of the `navbar` fields. `navbar: ~` means that the "Main verbs" will appear in the navbar without a heading; the absence of the `navbar` field in the developer vignettes means that they will only be accessible via the articles index.

The navbar will include a link to the articles index if one or more vignettes are not available through the navbar. If some vignettes appear in the navbar drop-down list and others do not, the list will automatically include a "More ..." link at the bottom; if no vignettes appear in the the navbar, it will link directly to the articles index instead of providing a drop-down.

Get started

Note that a vignette with the same name as the package (e.g., `vignettes/pkgdown.Rmd` or `vignettes/articles/pkgdown.Rmd`) automatically becomes a top-level "Get started" link, and will not appear in the articles drop-down.

(If your package name includes a `.`, e.g. `pack.down`, use a `-` in the vignette name, e.g. `pack-down.Rmd`.)

External files

pkgdown differs from base R in its handling of external files. When building vignettes, R assumes that vignettes are self-contained (a reasonable assumption when most vignettes were PDFs) and only copies files explicitly listed in `.install_extras`. pkgdown takes a different approach based on `rmarkdown::find_external_resources()`, and it will also copy any images that you link to. If for some reason the automatic detection doesn't work, you will need to add a `resource_files` field to the yaml metadata, e.g.:

```
---
title: My Document
resource_files:
  - data/mydata.csv
  - images/figure.png
---
```

Note that you can not use the `fig.path` to change the output directory of generated figures as its default value is a strong assumption of rmarkdown.

Embedding Shiny apps

If you would like to embed a Shiny app into an article, the app will have to be hosted independently, (e.g. <https://www.shinyapps.io>). Then, you can embed the app into your article using an `<iframe>`, e.g. `<iframe src = "https://gallery.shinyapps.io/083-front-page" class="shiny-app">`.

See <https://github.com/r-lib/pkgdown/issues/838#issuecomment-430473856> for some hints on how to customise the appearance with CSS.

Output formats

By default, pkgdown builds all articles using the `rmarkdown::html_document()` output format, ignoring whatever is set in your YAML metadata. This is necessary because pkgdown has to integrate the HTML/CSS/JS from the vignette with the HTML/CSS/JS from rest of the site. Because of the challenges of combining two sources of HTML/CSS/JS, there is limited support for other output formats and you have to opt-in by setting the `as_is` field in your `.Rmd` metadata:

```
pkgdown:
  as_is: true
```

If the output format produces a PDF, you'll also need to specify the extension field:

```
pkgdown:
  as_is: true
  extension: pdf
```

To work with pkgdown, the output format must accept `template`, `theme`, and `self_contained` arguments, and must work without any additional CSS or JSS files. Note that if you use `_output.yml` or `_site.yml` you'll still need to add `as_is: true` to each individual vignette.

Additionally, `htmlwidgets` do not work when `as_is: true`.

Suppressing vignettes

If you want articles that are not vignettes, either put them in subdirectories or list in `.Rbuildignore`. An articles link will be automatically added to the default navbar if the vignettes directory is present: if you do not want this, you will need to customise the navbar. See `build_site()` details.

Figures

You can control the default rendering of figures by specifying the `figures` field in `_pkgdown.yml`. The default settings are equivalent to:

```
figures:
  dev: ragg::agg_png
  dpi: 96
  dev.args: []
  fig.ext: png
  fig.width: 7.2916667
  fig.height: ~
  fig.retina: 2
  fig.asp: 1.618
  bg: NA
  other.parameters: []
```

Most of these parameters are interpreted similarly to knitr chunk options. `other.parameters` is a list of parameters that will be available to custom graphics output devices such as HTML widgets.

build_favicons	<i>Create favicons from package logo</i>
----------------	--

Description

This function auto-detects the location of your package logo (with the name `logo.svg` (recommended format) or `logo.png`) and runs it through the <https://realfavicongenerator.net> API to build a complete set of favicons with different sizes, as needed for modern web usage.

Usage

```
build_favicons(pkg = ".", overwrite = FALSE)
```

```
build_favicon(pkg, overwrite)
```

Arguments

<code>pkg</code>	Path to package.
<code>overwrite</code>	If TRUE, re-create favicons from package logo.

Details

You only need to run the function once. The favicon set will be stored in `pkgdown/favicon` and copied by `init_site()` to the relevant location when the website is rebuilt.

Once complete, you should add `pkgdown/` to `.Rbuildignore` to avoid a NOTE during package checking.

build_home_index	<i>Build home section</i>
------------------	---------------------------

Description

`build_home()` function generates pages at the top-level of the site including:

- The home page
- HTML files from any `.md` files in `./` or `.github/`.
- The authors page (from `DESCRIPTION`)
- The citation page (from `inst/CITATION`, if present).
- The license page
- A default 404 page if `.github/404.md` is not found.

`build_home_index()` rebuilds just the index page; it's useful for rapidly iterating when experimenting with site styles.

Usage

```
build_home_index(pkg = ".", quiet = TRUE)
```

```
build_home(pkg = ".", override = list(), preview = NA, quiet = TRUE)
```

Arguments

<code>pkg</code>	Path to package.
<code>quiet</code>	Set to <code>FALSE</code> to display output of knitr and pandoc. This is useful when debugging.
<code>override</code>	An optional named list used to temporarily override values in <code>_pkgdown.yml</code>
<code>preview</code>	If <code>TRUE</code> , or <code>is.na(preview) && interactive()</code> , will preview freshly generated section in browser.

Home page

The main content of the home page (`index.html`) is generated from `pkgdown/index.md`, `index.md`, or `README.md`, in that order. Most packages will use `README.md` because that's also displayed by GitHub and CRAN. Use `index.md` if you want your package website to look different to your `README`, and use `pkgdown/index.md` if you don't want that file to live in your package root directory.

If you use `index.Rmd` or `README.Rmd` it's your responsibility to knit the document to create the corresponding `.md`. `pkgdown` does not do this for you because it only touches files in the `doc/` directory.

Extra markdown files in the base directory (e.g. `ROADMAP.md`) or in `.github/` (e.g. `CODE_OF_CONDUCT.md`) are copied by `build_home()` to `docs/` and converted to HTML.

The home page also features a sidebar with information extracted from the package. You can tweak it via the configuration file, to help make the home page as informative as possible landing page.

Images and figures:

If you want to include images in your README.md, they must be stored somewhere in the package so that they can be displayed on the CRAN website. The best place to put them is `man/figures`. If you are generating figures with R Markdown, make sure you set up `fig.path` as followed:

```
knitr::opts_chunk$set(
  fig.path = "man/figures/"
)
```

This should usually go in a chunk with `include = FALSE`.

```
```{r chunk-name, include=FALSE}
knitr::opts_chunk$set(
 fig.path = "man/figures/"
)
```
```

Package logo:

If you have a package logo, you can include it at the top of your README in a level-one heading:

```
# pkgdown 
```

`[init_site()]` will also automatically create a favicon set from your package logo.

YAML config - home:

To tweak the home page, you need a section of the configuration file called `home`.

Page title and description:

By default, the page title and description are extracted automatically from the `Title` and `Description` fields `DESCRIPTION` (stripping single quotes off quoted words). CRAN ensures that these fields don't contain phrases like "R package" because that's obvious on CRAN. To make your package more findable on search engines, it's good practice to override the title and description, thinking about what people might search for:

```
home:
```

```
  title: An R package for pool-noodle discovery
```

```
  description: >
```

```
    Do you love R? Do you love pool-noodles? If so, you might enjoy
    using this package to automatically discover and add pool-noodles
    to your growing collection.
```

(Note the use of YAML's `>` i.e. "YAML pipes"; this is a convenient way of writing paragraphs of text.)

README header:

READMEs usually start with an `<h1>` containing the package name. If that feels duplicative with the package name in the navbar you can remove it with `strip_header: true`:

```
home:
```

```
  strip_header: true
```

Dev badges:

pkgdown identifies badges in three ways:

- Any image-containing links between `<!-- badges: start -->` and `<!-- badges: end -->`, as e.g. created by `usethis::use_readme_md()` or `usethis::use_readme_rmd()`. There should always be an empty line after the `<!-- badges: end -->` line. If you divide badges into paragraphs, make sure to add an empty line before the `<!-- badges: end -->` line.
- Any image-containing links within `<div id="badges"></div>`.
- Within the first paragraph, if it only contains image-containing links.

Authors

By default, pkgdown will display author information in three places:

- the sidebar,
- the left part side of the footer,
- the author page.

This documentation describes how to customise the overall author display. See `?build_home` and `?build_site` for details about changing the location of the authors information within the home sidebar and the site footer.

Authors ORCID and bio:

Author ORCID identification numbers in the DESCRIPTION are linked using the ORCID logo:

```
Authors@R: c(
  person("Hadley", "Wickham", , "hadley@rstudio.com", role = c("aut", "cre"),
    comment = c(ORCID = "0000-0003-4757-117X")
  ),
  person("Jay", "Hesselberth", role = "aut",
    comment = c(ORCID = "0000-0002-6299-179X")
  )
)
```

If you want to add more details about authors or their involvement with the package, you can use the comment field, which will be rendered on the authors page.

```
Authors@R: c(
  person("Hadley", "Wickham", , "hadley@rstudio.com", role = c("aut", "cre"),
    comment = c(ORCID = "0000-0003-4757-117X", "Indenter-in-chief")
  ),
  person("Jay", "Hesselberth", role = "aut",
    comment = c(ORCID = "0000-0002-6299-179X")
  )
)
```

YAML config - authors:

You can tweak a few things via the authors YAML field:

- display of each author in the footer, sidebar and authors page,
- which authors (by role) are displayed in the sidebar and footer,
- text before authors in the footer,
- text before and after authors in the sidebar,
- text before and after authors on the authors page.

You can modify how each author's name is displayed by adding a subsection for authors. Each entry in authors should be named with the author's name (matching DESCRIPTION) and can contain href and/or html fields:

- If href is provided, the author's name will be linked to this URL.
- If html is provided, it will be shown instead of the author's name. This is particularly useful if you want to display the logo of a corporate sponsor.

```
authors:
  firstname lastname:
    href: "http://name-website.com"
    html: "<img src='name-picture.png' width=72>"
```

By default, the "developers" list shown in the sidebar and footer is populated by the maintainer ("cre"), authors ("aut"), and funder ("fnd") from the DESCRIPTION. You could choose other roles for filtering. With the configuration below:

- only the maintainer and funder(s) appear in the footer, after the text "Crafted by",
- all authors and contributors appear in the sidebar,
- the authors list on the sidebar is preceded and followed by some text,
- the authors list on the authors page is preceded and followed by some text.

```
authors:
  footer:
    roles: [cre, fnd]
    text: "Crafted by"
  sidebar:
    roles: [aut, ctb]
    before: "So *who* does the work?"
    after: "Thanks all!"
    before: "This package is proudly brought to you by:"
    after: "See the [changelog](news/index.html) for other contributors. :pray:"
```

If you want to filter authors based on something else than their roles, consider using a custom sidebar/footer component (see ?build_home/?build_site, respectively).

Sidebar

You can customise the homepage sidebar with the `home.sidebar` field. It's made up of two pieces: `structure`, which defines the overall layout, and `components`, which defines what each piece looks like. This organisation makes it easy to mix and match the pkgdown defaults with your own customisations.

This is the default structure:

```
home:
  sidebar:
    structure: [links, license, community, citation, authors, dev]
```

These are drawn from seven built-in components:

- `links`: automated links generated from URL and BugReports fields from DESCRIPTION plus manual links from the `home.links` field:

```
home:
  links:
    - text: Link text
      href: https://website.com
    - text: Roadmap
      href: /roadmap.html
```

- `license`: Licensing information if LICENSE/LICENCE or LICENSE.md/LICENCE.md files are present.
- `community`: links to `.github/CONTRIBUTING.md`, `.github/CODE_OF_CONDUCT.md`, etc.
- `citation`: link to package citation information. Uses either `inst/CITATION` or, if absent, information from the DESCRIPTION.
- `authors`: selected authors from the DESCRIPTION.
- `dev`: development status badges found in README.md/index.md.

- toc: a table of contents for the README (not shown by default).

You can also add your own components, where text is markdown text:

```
home:
  sidebar:
    structure: [authors, custom, toc, dev]
    components:
      custom:
        title: Funding
        text: We are *grateful* for funding!
```

Alternatively, you can provide a ready-made sidebar HTML:

```
home:
  sidebar:
    html: path-to-sidebar.html
```

Or completely remove it:

```
home:
  sidebar: FALSE
```

build_news

Build news section

Description

A NEWS.md will be broken up into versions using level one (#) or level two headings (##) that (partially) match one of the following forms (ignoring case):

- {package name} 1.3.0
- {package name} v1.3.0
- Version 1.3.0
- Changes in 1.3.0
- Changes in v1.3.0

Usage

```
build_news(pkg = ".", override = list(), preview = NA)
```

Arguments

| | |
|----------|---|
| pkg | Path to package. |
| override | An optional named list used to temporarily override values in _pkgdown.yml |
| preview | If TRUE, or is.na(preview) && interactive(), will preview freshly generated section in browser. |

Details

A **common structure** for news files is to use a top level heading for each release, and use a second level heading to break up individual bullets into sections.

```
# foofy 1.0.0

## Major changes

* Can now work with all grooveable grobbles!

## Minor improvements and bug fixes

* Printing scrobbles no longer errors (@githubusername, #100)

* Wibbles are now 55% less jibbly (#200)
```

Issues and contributors will be automatically linked to the corresponding pages on GitHub if the GitHub repo can be discovered from the DESCRIPTION (typically from a URL entry containing github.com)

If a version is available on CRAN, the release date will automatically be added to the heading (see below for how to suppress); if not available on CRAN, "Unreleased" will be added.

YAML config

To automatically link to release announcements, include a releases section.

```
news:
  releases:
    - text: "usethis 1.3.0"
      href: https://www.tidyverse.org/articles/2018/02/usethis-1-3-0/
    - text: "usethis 1.0.0 (and 1.1.0)"
      href: https://www.tidyverse.org/articles/2017/11/usethis-1.0.0/
```

Control whether news is present on one page or multiple pages with the `one_page` field. The default is true.

```
news:
  one_page: false
```

Suppress the default addition of CRAN release dates with:

```
news:
  cran_dates: false
```

See Also

[Tidyverse style for News](#)

| | |
|-----------------|--------------------------------|
| build_reference | <i>Build reference section</i> |
|-----------------|--------------------------------|

Description

By default, pkgdown will generate an index that lists all functions in alphabetical order. To override this, provide a reference section in your `_pkgdown.yml` as described below.

Usage

```
build_reference(
  pkg = ".",
  lazy = TRUE,
  examples = TRUE,
  run_dont_run = FALSE,
  seed = 1014,
  override = list(),
  preview = NA,
  devel = TRUE,
  document = "DEPRECATED",
  topics = NULL
)

build_reference_index(pkg = ".")
```

Arguments

| | |
|--------------|---|
| pkg | Path to package. |
| lazy | If TRUE, only rebuild pages where the .Rd is more recent than the .html. This makes it much easier to rapidly prototype. It is set to FALSE by <code>build_site()</code> . |
| examples | Run examples? |
| run_dont_run | Run examples that are surrounded in <code>\dontrun</code> ? |
| seed | Seed used to initialize so that random examples are reproducible. |
| override | An optional named list used to temporarily override values in <code>_pkgdown.yml</code> |
| preview | If TRUE, or <code>is.na(preview) && interactive()</code> , will preview freshly generated section in browser. |
| devel | Determines how code is loaded in order to run examples. If TRUE (the default), assumes you are in a live development environment, and loads source package with <code>pkgload::load_all()</code> . If FALSE, uses the installed version of the package. |
| document | Deprecated Use <code>devel</code> instead. |
| topics | Build only specified topics. If supplied, sets <code>lazy</code> and <code>preview</code> to FALSE. |

Reference index

To tweak the index page, add a section called `reference` to `_pkgdown.yml`. It can contain three different types of element:

- A **title** (title + desc), which generates a row containing an `<h2>` with optional paragraph description.

- A **subtitle** (subtitle + desc), which generates an row containing an `<h3>` with optional paragraph description.
- A **list of topics** (contents), which generates one row for each topic, with a list of aliases for the topic on the left, and the topic title on the right.

(For historical reasons you can include contents with a title or subtitle, but this is no longer recommended).

Most packages will only need to use title and contents components. For example, here's a snippet from the YAML that pkgdown uses to generate its own reference index:

```
reference:
- title: Build
  desc: Build a complete site or its individual section components.
- contents:
  - starts_with("build_")
- title: Templates
- contents:
  - template_navbar
  - render_page
```

Bigger packages, e.g. `ggplot2`, may need an additional layer of structure in order to clearly organise large number of functions:

```
reference:
- title: Layers
- subtitle: Geoms
  desc: Geom is short for geometric element
- contents:
  - starts_with("geom")
- subtitle: Stats
  desc: Statistical transformations transform data before display.
  contents:
  - starts_with("stat")
```

desc can use markdown, and if you have a long description it's a good idea to take advantage of the YAML `>` notation:

```
desc: >
  This is a very _long_ and **overly** flowery description of a
  single simple function. By using `>`, it's easy to write a description
  that runs over multiple lines.
```

Topic matching:

contents can contain:

- Individual function/topic names.
- Weirdly named functions with doubled quoting, once for YAML and once for R, e.g. `"`+`".gg`"`.
- `starts_with("prefix")` to select all functions with common prefix.
- `ends_with("suffix")` to select all functions with common suffix.
- `matches("regex")` for more complex regular expressions.
- `has_keyword("x")` to select all topics with keyword "x"; `has_keyword("datasets")` selects all data documentation.

- `has_concept("blah")` to select all topics with concept "blah". If you are using roxygen2, `has_concept()` also matches family tags, because roxygen2 converts them to concept tags.
- `lacks_concepts(c("concept1", "concept2"))` to select all topics without those concepts. This is useful to capture topics not otherwise captured by `has_concepts()`.
- Topics from other installed packages, e.g. `rlang::is_installed()` (function name) or `sass::font_face` (topic name).

All functions (except for `has_keywords()`) automatically exclude internal topics (i.e. those with `\keyword{internal}`). You can choose to include with (e.g.) `starts_with("build_", internal = TRUE)`.

Use a leading `-` to remove topics from a section, e.g. `-topic_name`, `-starts_with("foo")`.

pkgdown will check that all non-internal topics are included on the reference index page, and will generate a warning if you have missed any. When the code is run in a continuous integration environment (i.e. when the CI envvar is "true"), this is an error so that the build will fail if you forget to include topics in the index.

Missing topics:

pkgdown will warn if there are (non-internal) topics that not listed in the reference index. You can suppress these warnings by listing the topics in section with "title: internal" (case sensitive) which will not be displayed on the reference index.

Icons:

You can optionally supply an icon for each help topic. To do so, you'll need a top-level icons directory. This should contain .png files that are either 30x30 (for regular display) or 60x60 (if you want retina display). Icons are matched to topics by aliases.

Examples:

If you need to run extra code before or after all examples are run, you can create `pkgdown/pre-reference.R` and `pkgdown/post-reference.R`.

Figures

You can control the default rendering of figures by specifying the `figures` field in `_pkgdown.yml`. The default settings are equivalent to:

```
figures:
  dev: ragg::agg_png
  dpi: 96
  dev.args: []
  fig.ext: png
  fig.width: 7.2916667
  fig.height: ~
  fig.retina: 2
  fig.asp: 1.618
  bg: NA
  other.parameters: []
```

Most of these parameters are interpreted similarly to knitr chunk options. `other.parameters` is a list of parameters that will be available to custom graphics output devices such as HTML widgets.

| | |
|--------------|---------------------------|
| build_search | <i>Build search index</i> |
|--------------|---------------------------|

Description

Build a JSON file encompassing all HTML pages, for use by the search script.

Usage

```
build_search(pkg = ".", override = list())
```

Arguments

| | |
|----------|---|
| pkg | Path to package. |
| override | An optional named list used to temporarily override values in <code>_pkgdown.yml</code> |

YAML config

You can exclude some paths from the search index. Below we exclude the changelog from the search index.

```
search:
  exclude: ['news/index.html']
```

Debugging and local testing

Locally (as opposed to on GitHub Pages or Netlify for instance), search won't work if you simply use pkgdown preview of the static files. You can use `servr::http("docs")` instead.

If search is not working, run `pkgdown::pkgdown_sitrep()` to eliminate common issues such as the absence of URL in the pkgdown configuration file of your package.

| | |
|------------|---|
| build_site | <i>Build a complete pkgdown website</i> |
|------------|---|

Description

`build_site()` is a convenient wrapper around six functions:

- `init_site()`
- `build_home()`
- `build_reference()`
- `build_articles()`
- `build_tutorials()`
- `build_news()`

See the documentation for the each function to learn how to control that aspect of the site. This page documents options that affect the whole site.

Usage

```
build_site(
  pkg = ".",
  examples = TRUE,
  run_dont_run = FALSE,
  seed = 1014,
  lazy = FALSE,
  override = list(),
  preview = NA,
  devel = FALSE,
  new_process = !devel,
  install = !devel,
  document = "DEPRECATED"
)
```

Arguments

| | |
|--------------|---|
| pkg | Path to package. |
| examples | Run examples? |
| run_dont_run | Run examples that are surrounded in \dontrun? |
| seed | Seed used to initialize so that random examples are reproducible. |
| lazy | If TRUE, will only rebuild articles and reference pages if the source is newer than the destination. |
| override | An optional named list used to temporarily override values in _pkgdown.yml |
| preview | If TRUE, or is.na(preview) && interactive(), will preview freshly generated section in browser. |
| devel | Use development or deployment process?

If TRUE, uses lighter-weight process suitable for rapid iteration; it will run examples and vignettes in the current process, and will load code with <code>pkgload::load_all()</code> .
If FALSE, will first install the package to a temporary library, and will run all examples and vignettes in a new process.

<code>build_site()</code> defaults to <code>devel = FALSE</code> so that you get high fidelity outputs when you building the complete site; <code>build_reference()</code> , <code>build_home()</code> and friends default to <code>devel = TRUE</code> so that you can rapidly iterate during development. |
| new_process | If TRUE, will run <code>build_site()</code> in a separate process. This enhances reproducibility by ensuring nothing that you have loaded in the current process affects the build process. |
| install | If TRUE, will install the package in a temporary library so it is available for vignettes. |
| document | Deprecated Use <code>devel</code> instead. |

General config

- `destination` controls where the site will be generated, defaulting to `docs/`. Paths are relative to the package root.
- `url` is optional, but strongly recommended.

url: <https://pkgdown.r-lib.org>

It specifies where the site will be published and is used to allow other pkgdown sites to link to your site when needed (`vignette("linking")`), generate a `sitemap.xml`, automatically generate a CNAME when [deploying to github](#), generate the metadata needed rich social "media cards" (`vignette("metadata")`), and more.

- `title` overrides the default site title, which is the package name. It's used in the page title and default navbar.

Development mode

The development field allows you to generate different sites for the development and released versions of your package. To use it, you first need to set the development mode:

```
development:
  mode: auto
```

Setting development mode:

The development mode of a site controls where the site is built, the colour of the package version in the navbar, the version tooltip, and whether or not the site is indexed by search engines. There are four possible modes:

- **automatic** (`mode: auto`): automatically determines the mode based on the version number:
 - 0.0.0.9000 (0.0.0.*): unreleased.
 - four version components: development.
 - everything else -> release.
- **release** (`mode: release`), the default. Site is written to `docs/`. Version in navbar gets the default colouring.
- **development** (`mode: devel`). Site is written to `docs/dev/`. The navbar version gets a "danger" class and a tooltip stating these are docs for an in-development version of the package. The `noindex` meta tag is used to ensure that these packages are not indexed by search engines.
- **unreleased** (`mode: unreleased`). Site is written to `docs/`. Version in navbar gets the "danger" class, and a message indicating the package is not yet on CRAN.

You can override the mode specified in the `_pkgdown.yml` by setting by setting `PKGDOWN_DEV_MODE` to `devel` or `release`.

Selective HTML:

You can selectively show HTML only on the devel or release site by adding class `pkgdown-devel` or `pkgdown-release`. This is most easily accessed from `.Rmd` files where you can use pandoc's `<div>` syntax to control where a block of markdown will display. For example, you can use the following markdown in your README to only show GitHub install instructions on the development version of your site:

```
::: {.pkgdown-devel}
```

You can install the development version of pkgdown from GitHub with:

```
`remotes::install_github("r-lib/pkgdown")`
```

```
:::
```

You can use a similar technique to control where badges are displayed. This markdown show the CRAN status badge on the site for the released package and the GitHub check status for the development package:

```
[![CRAN Status](https://www.r-pkg.org/badges/version/pkgdown)]
  (https://cran.r-project.org/package=pkgdown){.pkgdown-release}
[![R-CMD-check](https://github.com/r-lib/pkgdown/workflows/R-CMD-check/badge.svg)]
  (https://github.com/r-lib/pkgdown/actions){.pkgdown-devel}
```

Other options:

There are three other options that you can control:

```
development:
  destination: dev
  version_label: danger
  version_tooltip: "Custom message here"
```

`destination` allows you to override the default subdirectory used for the development site; it defaults to `dev/`. `version_label` allows you to override the style used for development (and unreleased) versions of the package. It defaults to "danger", but you can set to "default", "info", or "warning" instead. (The precise colours are determined by your bootstrap theme, but become progressively more eye catching as you go from default to danger). Finally, you can choose to override the default tooltip with `version_tooltip`.

Navigation bar

You can customise the navigation bar that appears at the top of the page with the `navbar` field. It's made up of two pieces: `structure`, which defines the overall layout, and `components`, which defines what each piece looks like. This organisation makes it easy to mix and match pkgdown defaults with your own customisations.

This is the default structure:

```
navbar:
  structure:
    left: [intro, reference, articles, tutorials, news]
    right: [search, github]
```

It makes use of the the six built-in components:

- `intro`: "Get Started", which links to a vignette with the same name as the package.
- `reference`, if there are any `.Rd` files.
- `articles`, if there are any vignettes or articles.
- `tutorials`, if there any tutorials.
- `news`, if `NEWS.md` exists.
- `search`, the search box (see `vignette("search")` for more details).
- `github`, a link to the source repository (with an icon), if it can be automatically determined from the `DESCRIPTION`.

You can use the `structure` field to reorganise the navbar without changing the default contents:

```
navbar:
  structure:
    left: [search]
    right: [reference, articles]
```

You can use `components` to override the default content. For example, this yaml provides a custom articles menu:

```
navbar:
  components:
    articles:
```

```

text: Articles
menu:
- text: Category A
- text: Title A1
  href: articles/a1.html
- text: Title A2
  href: articles/a2.html
- text: -----
- text: "Category B"
- text: Article B1
  href: articles/b1.html

```

Components uses the same syntax as **RMarkdown menus**. The elements of menu can be:

- A link (text + href)
- A heading (just text)
- A separator (text: -----)

Instead of text, you can also use the name of an icons from **fontawesome**. You should also provide a textual description in the `aria-label` field for screenreader users.

To add a new component to the navbar, you need to modify both structure and components. For example, the following yaml adds a new "twitter" component that appears to the left of the github icon.

```

navbar:
  structure:
    right: [twitter, github]
  components:
    twitter:
      icon: fa-twitter
      href: http://twitter.com/hadleywickham
      aria-label: Twitter

```

Finally, you can add arbitrary HTML to three locations in the navbar:

```

template:
  includes:
    before_title: <!-- inserted before the package title in the header -->
    before_navbar: <!-- inserted before the navbar links -->
    after_navbar: <!-- inserted after the navbar links -->

```

These includes will appear on all screen sizes, and will not be collapsed into the the navbar drop down.

You can also customise the colour scheme of the navbar by using the `type` and `bg` parameters. See `vignette("customise")` for more details.

Footer

You can customise the footer with the `footer` field. It's made up of two pieces: `structure`, which defines the overall layout, and `components`, which defines what each piece looks like. This organisation makes it easy to mix and match the pkgdown defaults with your own customisations.

This is the default structure:

```

footer:
  structure:
    left: developed_by
    right: built_with

```

Which uses two of the three built-in components:

- `developed_by`, a sentence describing the main authors of the package. (See `?build_home` if you want to tweak *which* authors appear in the footer.)
- `built_with`, a sentence advertising pkgdown.
- `package`, the name of the package.

You can override these defaults with the `footer` field. The example below puts the authors' information on the right along with a legal disclaimer, and puts the pkgdown link on the left.

```

footer:
  structure:
    left: pkgdown
    right: [authors, legal]
  components:
    legal: Provided without any warranty.

```

Each side is pasted into a single string (separated by " ") and then converted from markdown to HTML.

Search

The search field controls the built-in search. See `vignette("search")` for details.

Template

The `template` field is mostly used to control the appearance of the site. See `vignette("customise")` for details.

There are two other `template` fields that control other aspects of the site:

- `noindex: true` will suppress indexing of your pages by search engines:

```

template:
  params:
    noindex: true

```

- `google_site_verification` allows you to verify your site with google:

```

template:
  params:
    google_site_verification: _nn6ile-a6x6lct0W

```

- `trailing_slash_redirect: true` will automatically redirect `your-package-url.com` to `your-package-url.com/`, using a JS script added to the `<head>` of the home page. This is useful in certain redirect scenarios.

```

template:
  trailing_slash_redirect: true

```

Analytics

To capture usage of your site with a web analytics platform, you can make use of the `includes` field to add the HTML supplied to you by the platform. Typically these are either placed after `_body` or `in_header`. I include a few examples below, but I highly recommend getting the recommended HTML directly from the platform.

- **counter.dev**

```
template:
  includes:
    after_body: >
      <script>if(!sessionStorage.getItem("_swa")&&document.referrer.indexOf(location.protocol) !== -1){
        fetch("https://counter.dev/track?" + new URLSearchParams({referrer:document.referrer,screensize:screen.width+"x"+screen.height,utcoffset:"1"}));sessionStorage.setItem("_swa","1");</script>
```

- **GoatCounter:**

```
template:
  includes:
    after_body: >
      <script data-goatcounter="https://{YOUR CODE}.goatcounter.com/count" data-goatcounter-se-
```

- **Google analytics:**

```
template:
  includes:
    in_header: |
      <!-- Global site tag (gtag.js) - Google Analytics -->
      <script async src="https://www.googletagmanager.com/gtag/js?id={YOUR TRACKING ID}"# ' ></script>
      <script>
        window.dataLayer = window.dataLayer || [];
        function gtag(){dataLayer.push(arguments);}
        gtag('js', new Date());

        gtag('config', '{YOUR TRACKING ID}');
      </script>
```

- **plausible.io:**

```
templates:
  includes:
    in_header: |
      <script defer data-domain="{YOUR DOMAIN}" src="https://plausible.io/js/plausible.js"></script>
```

Source repository

Use the `repo` field to override pkgdown's automatic discovery of your source repository. This is used in the navbar, on the homepage, in articles and reference topics, and in the changelog (to link to issue numbers and user names). pkgdown can automatically figure out the necessary URLs if you link to a GitHub or GitLab repo in your BugReports or URL field.

Otherwise, you can supply your own in the `repo` field:

```
repo:
  url:
    home: https://github.com/r-lib/pkgdown/
    source: https://github.com/r-lib/pkgdown/blob/HEAD/
```

```

issue: https://github.com/r-lib/pkgdown/issues/
user: https://github.com/

```

- home: path to package home on source code repository.
- source:: path to source of individual file in default branch.
- issue: path to individual issue.
- user: path to user.

The varying components (e.g. path, issue number, user name) are pasted on the end of these URLs so they should have trailing /s.

pkgdown can automatically link to Jira issues as well if specify both a custom issue URL as well Jira project names to auto-link in `jira_projects`. You can specify as many projects as you would like:

```

repo:
  jira_projects: [this_project, another_project]
  url:
    issue: https://jira.organisation.com/jira/browse/

```

pkgdown defaults to using the "HEAD" branch for source file URLs. This can be configured to use a specific branch when linking to source files by specifying a branch name:

```

repo:
  branch: devel

```

Deployment (deploy)

There is a single deploy field

- `install_metadata` allows you to install package index metadata into the package itself. Normally this metadata is made available on the published site; installing it into your package means that it's available for autolinking even if your website is not reachable at build time (e.g. because behind a firewall or requires auth).

```

deploy:
  install_metadata: true

```

Redirects

If you change the structure of your documentation (by renaming vignettes or help topics) you can setup redirects from the old content to the new content. One or several now-absent pages can be redirected to a new page (or to a new section of a new page). This works by creating a html page that performs a "meta refresh", which isn't the best way of doing a redirect but works everywhere that you might deploy your site.

The syntax is the following, with old paths on the left, and new paths or URLs on the right.

```

redirects:
- ["articles/old-vignette-name.html", "articles/new-vignette-name.html"]
- ["articles/another-old-vignette-name.html", "articles/new-vignette-name.html"]
- ["articles/yet-another-old-vignette-name.html", "https://pkgdown.r-lib.org/dev"]

```

If for some reason you choose to redirect an existing page make sure to exclude it from the search index, see `?build_search`.

Options

Users with limited internet connectivity can disable CRAN checks by setting `options(pkgdown.internet = FALSE)`. This will also disable some features from pkgdown that requires an internet connectivity. However, if it is used to build docs for a package that requires internet connectivity in examples or vignettes, this connection is required as this option won't apply on them.

Users can set a timeout for `build_site(new_process = TRUE)` with `options(pkgdown.timeout = Inf)`, which is useful to prevent stalled builds from hanging in cron jobs.

Examples

```
## Not run:
build_site()

build_site(override = list(destination = tempdir()))

## End(Not run)
```

`build_site_github_pages`

Build site for GitHub pages

Description

Designed to be run as part of automated workflows for deploying to GitHub pages. It cleans out the old site, builds the site into `dest_dir` adds a `.nojekyll` file to suppress rendering by Jekyll, and adds a CNAME file if needed.

It is designed to be run in CI, so by default it:

- Clean out the old site.
- Does not install the package.
- Runs `build_site()` in process.

Usage

```
build_site_github_pages(
  pkg = ".",
  ...,
  dest_dir = "docs",
  clean = TRUE,
  install = FALSE,
  new_process = FALSE
)
```

Arguments

| | |
|-----------------------|--|
| <code>pkg</code> | Path to package. |
| <code>...</code> | Additional arguments passed to <code>build_site()</code> . |
| <code>dest_dir</code> | Directory to build site in. |
| <code>clean</code> | Clean all files from old site. |

| | |
|-------------|--|
| install | If TRUE, will install the package in a temporary library so it is available for vignettes. |
| new_process | If TRUE, will run build_site() in a separate process. This enhances reproducibility by ensuring nothing that you have loaded in the current process affects the build process. |

| | |
|-----------------|--------------------------------|
| build_tutorials | <i>Build tutorials section</i> |
|-----------------|--------------------------------|

Description

learnr tutorials must be hosted elsewhere as they require an R execution engine. Currently, pkgdown will not build or publish tutorials for you, but makes it easy to embed (using `<iframe>`s) published tutorials. Tutorials are automatically discovered from published tutorials in `inst/tutorials` and `vignettes/tutorials`. Alternatively, you can list in `_pkgdown.yml` as described below.

Usage

```
build_tutorials(pkg = ".", override = list(), preview = NA)
```

Arguments

| | |
|----------|---|
| pkg | Path to package. |
| override | An optional named list used to temporarily override values in <code>_pkgdown.yml</code> |
| preview | If TRUE, or <code>is.na(preview) && interactive()</code> , will preview freshly generated section in browser. |

YAML config

To override the default discovery process, you can provide a `tutorials` section. This should be a list where each element specifies:

- `name`: used for the generated file name
- `title`: used in page heading and in navbar
- `url`: which will be embedded in an `iframe`
- `source`: optional, but if present will be linked to

```
tutorials:
- name: 00-setup
  title: Setting up R
  url: https://jjallaire.shinyapps.io/learnr-tutorial-00-setup/
- name: 01-data-basics
  title: Data basics
  url: https://jjallaire.shinyapps.io/learnr-tutorial-01-data-basics/
```

| | |
|------------|-------------------|
| clean_site | <i>Clean site</i> |
|------------|-------------------|

Description

Delete all files in docs/ (except for CNAME).

Usage

```
clean_site(pkg = ".")
```

Arguments

| | |
|-----|------------------|
| pkg | Path to package. |
|-----|------------------|

| | |
|------------------|--|
| deploy_to_branch | <i>Build and deploy a site locally</i> |
|------------------|--|

Description

Assumes that you're in a git clone of the project, and the package is already installed. Use [usethis::use_pkgdown_github_actions\(\)](#) to automate this process using GitHub actions.

Usage

```
deploy_to_branch(
  pkg = ".",
  commit_message = construct_commit_message(pkg),
  clean = TRUE,
  branch = "gh-pages",
  remote = "origin",
  github_pages = (branch == "gh-pages"),
  ...
)
```

Arguments

| | |
|----------------|--|
| pkg | Path to package. |
| commit_message | The commit message to be used for the commit. |
| clean | Clean all files from old site. |
| branch | The git branch to deploy to |
| remote | The git remote to deploy to |
| github_pages | Is this a GitHub pages deploy. If TRUE, adds a CNAME file for custom domain name support, and a .nojekyll file to suppress jekyll rendering. |
| ... | Additional arguments passed to build_site() . |

| | |
|-----------|---------------------------------------|
| init_site | <i>Initialise site infrastructure</i> |
|-----------|---------------------------------------|

Description

init_site():

- creates the output directory (docs/),
- generates a machine readable description of the site, used for autolinking,
- copies CSS/JS assets and extra files, and
- runs build_favicons(), if needed.

See vignette("customise") for the various ways you can customise the display of your site.

Usage

```
init_site(pkg = ".")
```

Arguments

pkg Path to package.

Build-ignored files

We recommend using `usethis::use_pkgdown()` to build-ignore docs/ and _pkgdown.yml. If use another directory, or create the site manually, you'll need to add them to .Rbuildignore yourself. A NOTE about an unexpected file during R CMD CHECK is an indication you have not correctly ignored these files.

| | |
|------------|---|
| in_pkgdown | <i>Determine if code is executed by pkgdown</i> |
|------------|---|

Description

This is occasionally useful when you need different behaviour by pkgdown and regular documentation.

Usage

```
in_pkgdown()
```

Examples

```
in_pkgdown()
```

| | |
|----------------|---|
| pkgdown_sitrep | <i>Report package pkgdown situation</i> |
|----------------|---|

Description

pkgdown_sitrep() reports

- If there is an url field in the pkgdown configuration;
- If that pkgdown website URL is stored in the DESCRIPTION file.

Usage

```
pkgdown_sitrep(pkg = ".")
```

Arguments

| | |
|-----|------------------|
| pkg | Path to package. |
|-----|------------------|

| | |
|--------------|-----------------------------|
| preview_site | <i>Open site in browser</i> |
|--------------|-----------------------------|

Description

Open site in browser

Usage

```
preview_site(pkg = ".", path = ".", preview = NA)
```

Arguments

| | |
|---------|---|
| pkg | Path to package. |
| path | Path relative to destination |
| preview | If TRUE, or is.na(preview) && interactive(), will preview freshly generated section in browser. |

| | |
|---------|--|
| rd2html | <i>Translate an Rd string to its HTML output</i> |
|---------|--|

Description

Translate an Rd string to its HTML output

Usage

```
rd2html(x, fragment = TRUE, ...)
```

Arguments

| | |
|----------|---|
| x | Rd string. Backslashes must be double-escaped ("\\"). |
| fragment | logical indicating whether this represents a complete Rd file |
| ... | additional arguments for as_html |

Examples

```
rd2html("a\n%b\nc")

rd2html("a & b")

rd2html("\\strong{\\emph{x}}")
```

| | |
|-------------|----------------------------------|
| render_page | <i>Render page with template</i> |
|-------------|----------------------------------|

Description

Each page is composed of four templates: "head", "header", "content", and "footer". Each of these templates is rendered using the data, and then assembled into an overall page using the "layout" template.

Usage

```
render_page(pkg = ".", name, data, path = "", depth = NULL, quiet = FALSE)

data_template(pkg = ".", depth = 0L)
```

Arguments

| | |
|------|--|
| pkg | Path to package to document. |
| name | Name of the template (e.g. "home", "vignette", "news") |
| data | Data for the template.
This is automatically supplemented with three lists: <ul style="list-style-type: none"> • site: title and path to root. |

- yaml: the template key from _pkgdown.yml.
- package: package metadata including name and version.

See the full contents by running [data_template\(\)](#).

| | |
|-------|--|
| path | Location to create file; relative to destination directory. If "" (the default), prints to standard out. |
| depth | Depth of path relative to base directory. |
| quiet | If quiet, will suppress output messages |

| | |
|-----------------|--------------------------------|
| template_navbar | <i>Generate YAML templates</i> |
|-----------------|--------------------------------|

Description

Use these function to generate the default YAML that pkgdown uses for the different parts of _pkgdown.yml. These are useful starting points if you want to customise your site.

Usage

```
template_navbar(path = ".")
```

```
template_reference(path = ".")
```

```
template_articles(path = ".")
```

Arguments

| | |
|------|----------------------|
| path | Path to package root |
|------|----------------------|

Examples

```
## Not run:
pkgdown::template_navbar()

## End(Not run)

## Not run:
pkgdown::template_reference()

## End(Not run)

## Not run:
pkgdown::template_articles()

## End(Not run)
```