

Package ‘blogdown’

October 11, 2020

Type Package

Title Create Blogs and Websites with R Markdown

Version 0.21

Description Write blog posts and web pages in R Markdown. This package supports the static site generator 'Hugo' (<<https://gohugo.io>>) best, and it also supports 'Jekyll' (<<https://jekyllrb.com>>) and 'Hexo' (<<https://hexo.io>>).

License GPL-3

Imports rmarkdown (>= 2.4), bookdown (>= 0.14), knitr (>= 1.25),
htmltools, yaml (>= 2.1.19), httpuv (>= 1.4.0), later, xfun (>= 0.18),
servr (>= 0.19)

Suggests testit, shiny, miniUI, stringr, rstudioapi, tools, processx,
whoami

LazyData true

URL <https://github.com/rstudio/blogdown>

BugReports <https://github.com/rstudio/blogdown/issues>

SystemRequirements Hugo (<<https://gohugo.io>>) and Pandoc
(<<https://pandoc.org>>)

RoxygenNote 7.1.1

Encoding UTF-8

NeedsCompilation no

Author Yihui Xie [aut, cre] (<<https://orcid.org/0000-0003-0645-5666>>),
Beilei Bian [ctb],
Forest Fang [ctb],
Garrick Aden-Buie [ctb],
Hiroaki Yutani [ctb],
Ian Lyttle [ctb],
JJ Allaire [ctb],
Kevin Ushey [ctb],
Leonardo Collado-Torres [ctb],
Xianying Tan [ctb],
Raniere Silva [ctb],

Jozef Hajnala [ctb],
RStudio, PBC [cph]
Maintainer Yihui Xie <xie@yihui.name>
Repository CRAN
Date/Publication 2020-10-11 07:20:02 UTC

R topics documented:

blogdown	2
build_dir	3
build_site	3
bundle_site	4
dep_path	5
find_yaml	6
html_page	7
hugo_cmd	8
install_hugo	11
install_theme	12
md5sum_filter	13
read_toml	14
serve_site	15
shortcode	16
Index	18

blogdown	<i>The blogdown package</i>
----------	------------------------------------

Description

The comprehensive documentation of this package is the book **blogdown: Creating Websites with R Markdown** (<https://bookdown.org/yihui/blogdown/>). You are expected to read at least the first chapter. If you are really busy or do not care about an introduction to **blogdown** (e.g., you are very familiar with creating websites), set your working directory to an empty directory, and run `blogdown::new_site()` to get started right away.

Examples

```
if (interactive()) blogdown::new_site()
```

build_dir	<i>Build all Rmd files under a directory</i>
-----------	--

Description

List all Rmd files recursively under a directory, and compile them using `rmarkdown::render()`.

Usage

```
build_dir(dir = ".", force = FALSE, ignore = "[.]Rproj$")
```

Arguments

dir	A directory path.
force	Whether to force building all Rmd files. By default, an Rmd file is built only if it is newer than its output file(s).
ignore	A regular expression to match output filenames that should be ignored when testing if the modification time of the Rmd source file is newer than its output files.

build_site	<i>Build a website</i>
------------	------------------------

Description

Build the site through Hugo, and optionally (re)build R Markdown files.

Usage

```
build_site(
  local = FALSE,
  method = c("html", "custom"),
  run_hugo = TRUE,
  build_rmd = FALSE
)
```

Arguments

local	Whether to build the website locally. This argument is passed to <code>hugo_build()</code> , and <code>local = TRUE</code> is mainly for serving the site locally via <code>serve_site()</code> .
method	Different methods to build a website (each with pros and cons). See ‘Details’. The value of this argument will be obtained from the global option <code>getOption('blogdown.method')</code> when it is set.
run_hugo	Whether to run <code>hugo_build()</code> after R Markdown files are compiled.
build_rmd	Whether to (re)build R Markdown files. By default, they are not rebuilt. See ‘Details’ for how <code>build_rmd = TRUE</code> works. Alternatively, it can take a vector of file paths, which means these files are to be (re)built.

Details

You can use `serve_site()` to preview your website locally, and `build_site()` to build the site for publishing. However, if you use a web publishing service like Netlify, you do not need to build the site locally, but can build it on the cloud. See Section 1.7 of the **blogdown** book for more information: <https://bookdown.org/yihui/blogdown/workflow.html>.

For the method argument: `method = "html"` means to render Rmd files to HTML via `rmarkdown::render()` (which means Markdown is processed through Pandoc), and process the paths of external dependencies generated from R code chunks, including images and HTML dependencies.

For all rendering methods, a custom R script 'R/build.R' will be executed if you have provided it under the root directory of the website (e.g. you can compile Rmd to Markdown through `knitr::knit()` and build the site via `hugo_cmd()`). `method = "custom"` means it is entirely up to this R script how a website is rendered. The script is executed via command line Rscript "R/build.R", which means it is executed in a separate R session. The value of the argument `local` is passed to the command line (you can retrieve the command-line arguments via `commandArgs(TRUE)`). For `method = "html"`, the R script 'R/build2.R' (if exists) will be executed after Hugo has built the site. This can be useful if you want to post-process the site.

When `build_rmd = TRUE`, all Rmd files will be (re)built. You can set the global option `blogdown.files_filter` to a function to determine which Rmd files to build when `build_rmd = TRUE`. This function takes a vector of Rmd file paths, and should return a subset of these paths to be built. By default, `options(blogdown.files_filter = identity)`. You can use `blogdown::timestamp_filter`, which filters the files by comparing their time stamps (modification time) with the time stamps of their output files. Another filter function is `blogdown::md5sum_filter`, which is more robust in determining if an Rmd file has been modified (hence needs to be rebuilt).

bundle_site

Convert post files to leaf bundles

Description

For a post with the path 'content/path/to/my-post.md', it will be moved to 'content/path/to/my-post/index.md', so it becomes the index file of a leaf bundle of Hugo. This also applies to files with extensions '.Rmd' and '.Rmarkdown'.

Usage

```
bundle_site(dir = site_root(), output)
```

Arguments

<code>dir</code>	The root directory of the website project (should contain a 'content/' folder).
<code>output</code>	The output directory. If not provided, a suffix '-bundle' is added to the website root directory name. For example, the default output directory for the site under '~/Documents/test' is '~/Documents/test-bundle'. You can specify the output directory to be identical to the website root directory, so files will be moved within the same directory, but please remember that you will not be able to undo <code>bundle_site()</code> . You should modify the website in place <i>only if you have a backup for this directory or it is under version control</i> .

Note

This function only moves (R) Markdown source files. If these files use resource files under the ‘static/’ folder, these resources will not be moved into the ‘content/’ folder. You need to manually move them, and adjust their paths in the (R) Markdown source files accordingly.

References

Learn more about Hugo’s leaf bundles at <https://gohugo.io/content-management/page-bundles/>.

Examples

```
## Not run:
blogdown::bundle_site(".", "../new-site/")
blogdown::bundle_site(".", ".") # move files within the current working directory

## End(Not run)
```

dep_path

A helper function to return a dependency path name

Description

In most cases, **blogdown** can process images and HTML widgets automatically generated from code chunks (they will be moved to the `static/` folder by default), but it may fail to recognize dependency files generated to other paths. This function returns a path that you can use for your output files, so that **blogdown** knows that they should be processed, too. It is designed to be used in a **knitr** code chunk.

Usage

```
dep_path(default = knitr::opts_chunk$get("fig.path"))
```

Arguments

default	Return this default value when this function is called outside of a knitr code chunk.
---------	--

Value

A character string of the default value (outside **knitr**), or a path consisting of the **knitr** figure path appended by the current chunk label.

`find_yaml`*Find posts containing the specified metadata*

Description

Given a YAML field name, find the (R) Markdown files that contain this field and its value contains any of the specified values. Functions `find_tags()` and `find_categories()` are wrappers of `find_yaml()` with `field = 'tags'` and `field = 'categories'`, respectively; `count_fields()` returns the frequency tables of the specified YAML fields, such as the counts of tags and categories.

Usage

```
find_yaml(field = character(), value = character(), open = FALSE)
```

```
find_tags(value = character(), open = FALSE)
```

```
find_categories(value = character(), open = FALSE)
```

```
count_yaml(fields = c("categories", "tags"), sort_by_count = TRUE)
```

Arguments

<code>field, fields</code>	A character vector of YAML field names.
<code>value</code>	A vector of the field values to be matched.
<code>open</code>	Whether to open the matched files automatically.
<code>sort_by_count</code>	Whether to sort the frequency tables by counts.

Value

`find_yaml()` returns a character vector of filenames; `count_yaml()` returns a list of frequency tables.

Examples

```
library(blogdown)
find_tags(c("time-series", "support vector machine"))
find_categories("Statistics")

count_yaml(sort_by_count = FALSE)
```

Description

This function is a simple wrapper of `bookdown::html_document2()` with different default arguments, and more importantly, a special HTML template designed only for **blogdown** to render R Markdown to HTML pages that can be processed by Hugo.

Usage

```
html_page(  
  ...,  
  number_sections = FALSE,  
  self_contained = FALSE,  
  highlight = NULL,  
  template = NULL,  
  keep_md = FALSE,  
  pre_knit = NULL,  
  post_processor = NULL  
)
```

Arguments

`..., number_sections, self_contained, highlight, template`
Arguments passed to `bookdown::html_document2()` (note the option `theme` is not supported and set to `NULL` internally, and when `template = NULL`, a default template in **blogdown** will be used).

`keep_md, pre_knit, post_processor`
Passed to `rmarkdown::output_format`.

Details

The HTML output is not a complete HTML document, and only meaningful to **blogdown** (it will be post-processed to render valid HTML pages). The only purpose of this output format is for users to change options in YAML.

The fact that it is based on **bookdown** means most **bookdown** features are supported, such as numbering and cross-referencing figures/tables.

Note

Do not use a custom template unless you understand how the default template actually works (see the **blogdown** book).

The argument `highlight` does not support the value `"textmate"`, and the argument `template` does not support the value `"default"`.

References

See Chapter 2 of the **bookdown** book for the Markdown syntax: <https://bookdown.org/yihui/bookdown>. See the **blogdown** book for full details: <https://bookdown.org/yihui/blogdown>.

hugo_cmd

Run Hugo commands

Description

Wrapper functions to run Hugo commands via `system2('hugo', ...)`.

Usage

```
hugo_cmd(...)

hugo_version()

hugo_available(version = "0.0.0")

hugo_build(local = FALSE)

new_site(
  dir = ".",
  install_hugo = TRUE,
  format = "toml",
  sample = TRUE,
  theme = "yihui/hugo-lithium",
  hostname = "github.com",
  theme_example = TRUE,
  empty_dirs = FALSE,
  to_yaml = TRUE,
  serve = interactive()
)

new_content(path, kind = "", open = interactive())

new_post(
  title,
  kind = "",
  open = interactive(),
  author = getOption("blogdown.author"),
  categories = NULL,
  tags = NULL,
  date = Sys.Date(),
  file = NULL,
  slug = NULL,
```

```

    title_case = getOption("blogdown.title_case"),
    subdir = getOption("blogdown.subdir", "post"),
    ext = getOption("blogdown.ext", ".md")
  )

  hugo_convert(to = c("YAML", "TOML", "JSON"), unsafe = FALSE, ...)

  hugo_server(host, port)

```

Arguments

...	Arguments to be passed to <code>system2('hugo', ...)</code> , e.g. <code>new_content(path)</code> is basically <code>hugo_cmd(c('new', path))</code> (i.e. run the command <code>hugo new path</code>).
version	A version number.
local	Whether to build the site for local preview (if TRUE, all drafts and future posts will also be built, and the site configuration <code>baseurl</code> will be set to / temporarily).
dir	The directory of the new site. It should be empty or only contain hidden files, RStudio project (<code>*.Rproj</code>) files, <code>'LICENSE'</code> , and/or <code>'README'/'README.md'</code> .
install_hugo	Whether to install Hugo automatically if it is not found.
format	The format of the configuration file. Note that the frontmatter of the new (R) Markdown file created by <code>new_content()</code> always uses YAML instead of TOML.
sample	Whether to add sample content. Hugo creates an empty site by default, but this function adds sample content by default.
theme	A Hugo theme on Github (a character string of the form <code>user/repo</code> , and you can optionally specify a GIT branch or tag name after @, i.e. theme can be of the form <code>user/repo@branch</code>). You can also specify a full URL to the zip file or tarball of the theme. If <code>theme = NA</code> , no themes will be installed, and you have to manually install a theme.
hostname	Where to find the theme. Defaults to <code>github.com</code> ; specify if you wish to use an instance of GitHub Enterprise. You can also specify the full URL of the zip file or tarball in theme, in which case this argument is ignored.
theme_example	Whether to copy the example in the <code>'exampleSite'</code> directory if it exists in the theme. Not all themes provide example sites.
empty_dirs	Whether to keep the empty directories generated by Hugo.
to_yaml	Whether to convert the metadata of all posts to YAML.
serve	Whether to start a local server to serve the site.
path	The path to the new file under the <code>'content'</code> directory.
kind	The content type to create, i.e., the Hugo archetype. If the archetype is a page bundle archetype, it should end with a slash, e.g., <code>post/</code> .
open	Whether to open the new file after creating it. By default, it is opened in an interactive R session.
title	The title of the post.
author	The author of the post.

categories	A character vector of category names.
tags	A character vector of tag names.
date	The date of the post.
file	The filename of the post. By default, the filename will be automatically generated from the title by replacing non-alphanumeric characters with dashes, e.g. title = 'Hello World' may create a file 'content/post/2016-12-28-hello-world.md'. The date of the form YYYY-mm-dd will be prepended if the filename does not start with a date.
slug	The slug of the post. By default (NULL), the slug is generated from the filename by removing the date and filename extension, e.g., if file = 'post/2015-07-23-hi-there.md', slug will be hi-there. Set slug = '' if you do not want it.
title_case	A function to convert the title to title case. If TRUE, the function is <code>tools::toTitleCase()</code> . This argument is not limited to title case conversion. You can provide an arbitrary R function to convert the title.
subdir	If specified (not NULL), the post will be generated under a subdirectory under 'content/'. It can be a nested subdirectory like 'post/joe/'.
ext	The filename extension (e.g., '.md', '.Rmd', or '.Rmarkdown'). Ignored if file has been specified.
to	A format to convert to.
unsafe	Whether to enable unsafe operations, such as overwriting Markdown source documents. If you have backed up the website, or the website is under version control, you may try unsafe = TRUE.
host, port	The host IP address and port; see <code>servr::server_config()</code> .

Functions

- `hugo_cmd`: Run an arbitrary Hugo command.
- `hugo_version`: Return the version number of Hugo if possible, which is extracted from the output of `hugo_cmd('version')`.
- `hugo_available`: Check if Hugo with a certain version or above is available.
- `hugo_build`: Build a plain Hugo website. Note that the function `build_site()` first compiles Rmd files, and then calls Hugo via `hugo_build()` to build the site.
- `new_site`: Create a new site (skeleton) via hugo new site. The directory of the new site should be empty,
- `new_content`: Create a new (R) Markdown file via hugo new (e.g. a post or a page).
- `new_post`: A wrapper function to create a new post under the 'content/post/' directory via `new_content()`. If your post will use R code chunks, you can set `ext = '.Rmd'` or the global option `options(blogdown.ext = '.Rmd')` in your '~/.Rprofile'. Similarly, you can set `options(blogdown.author = 'Your Name')` so that the author field is automatically filled out when creating a new post.
- `hugo_convert`: A wrapper function to convert source content to different formats via `hugo convert`.
- `hugo_server`: Start a Hugo server.

References

The full list of Hugo commands: <https://gohugo.io/commands>, and themes: <https://themes.gohugo.io>.

Examples

```
blogdown::hugo_available("1.2.3")
if (interactive()) blogdown::new_site()
```

install_hugo	<i>Install Hugo</i>
--------------	---------------------

Description

Download the appropriate Hugo executable for your platform from Github and try to copy it to a system directory so **blogdown** can run the hugo command to build a site. `update_hugo()` is a wrapper of `install_hugo(force = TRUE)`.

Usage

```
install_hugo(
  version = "latest",
  use_brew = Sys.which("brew") != "",
  force = FALSE,
  extended = TRUE
)

update_hugo()
```

Arguments

version	The Hugo version number, e.g., 0.26; the special value latest means the latest version (fetched from Github releases). Alternatively, this argument can take a file path of the zip archive or tarball of the Hugo installer that has already been downloaded from Github, in which case it will not be downloaded again.
use_brew	Whether to use Homebrew (https://brew.sh) on macOS to install Hugo (recommended if you have already installed Homebrew). Note Homebrew will be automatically installed if it has not been installed when <code>use_brew = TRUE</code> .
force	Whether to install Hugo even if it has already been installed. This may be useful when upgrading Hugo (if you use Homebrew, run the command <code>brew update && brew upgrade</code> instead).
extended	Whether to use extended version of Hugo that has SCSS/SASS support. You only need the extended version if you want to edit SCSS/SASS.

Details

This function tries to install Hugo to `Sys.getenv('APPDATA')` on Windows, `~/Library/Application Support` on macOS, and `~/bin/` on other platforms (such as Linux). If these directories are not writable, the package directory 'Hugo' of **blogdown** will be used. If it still fails, you have to install Hugo by yourself and make sure it can be found via the environment variable `PATH`.

This is just a helper function and may fail to choose the correct Hugo executable for your operating system, especially if you are not on Windows or Mac or a major Linux distribution. When in doubt, read the Hugo documentation and install it by yourself: <https://gohugo.io>.

If you want to install Hugo to a custom path, you can set the global option `blogdown.hugo.dir` to a directory to store the Hugo executable before you call `install_hugo()`, e.g., `options(blogdown.hugo.dir = '~/Downloads/hugo_0.20.1/')`. This may be useful for you to use a specific version of Hugo for a specific website. You can set this option per project. See [Section 1.4 Global options](#) for details, or store a copy of Hugo on a USB Flash drive along with your website.

install_theme

Install a Hugo theme from Github

Description

Download the specified theme from Github and install to the 'themes' directory. Available themes are listed at <https://themes.gohugo.io>.

Usage

```
install_theme(
  theme,
  hostname = "github.com",
  theme_example = FALSE,
  update_config = TRUE,
  force = FALSE,
  update_hugo = TRUE
)
```

Arguments

theme	A Hugo theme on Github (a character string of the form <code>user/repo</code> , and you can optionally specify a GIT branch or tag name after <code>@</code> , i.e. <code>theme</code> can be of the form <code>user/repo@branch</code>). You can also specify a full URL to the zip file or tarball of the theme. If <code>theme = NA</code> , no themes will be installed, and you have to manually install a theme.
hostname	Where to find the theme. Defaults to <code>github.com</code> ; specify if you wish to use an instance of GitHub Enterprise. You can also specify the full URL of the zip file or tarball in <code>theme</code> , in which case this argument is ignored.
theme_example	Whether to copy the example in the 'exampleSite' directory if it exists in the theme. Not all themes provide example sites.

update_config	Whether to update the theme option in the site configurations.
force	Whether to override the existing theme of the same name. If you have made changes to this existing theme, your changes will be lost when force = TRUE! Please consider backing up the theme by renaming it before you try force = TRUE.
update_hugo	Whether to automatically update Hugo if the theme requires a higher version of Hugo than the existing version in your system.

md5sum_filter	<i>Look for files that have been possibly modified</i>
---------------	--

Description

Filter files by checking if their modification times or MD5 checksums have changed.

Usage

```
md5sum_filter(files, db = "blogdown/md5sum.txt")

timestamp_filter(files)
```

Arguments

files	A vector of file paths.
db	Path to the database file.

Details

The function `md5sum_filter()` reads the MD5 checksums of files from a database (a tab-separated text file), and returns the files of which the checksums have changed. If the database does not exist, write the checksums of files to it, otherwise update the checksums after the changed files have been identified. When a file is modified, its MD5 checksum is very likely to change.

The function `timestamp_filer()` compares the modification time of an Rmd file with that of its output file, and returns a file if it's newer than its output file by N seconds (or if the output file does not exist), where N is obtained from the R global option `blogdown.time_diff`. By default, `N = 0`. You may change it via `options()`, e.g., `options(blogdown.time_diff = 5)` means an Rmd file will be returned when its modification time at least 5 seconds newer than its output file's modification time.

These functions can be used to determine which Rmd files to be rebuilt in a **blogdown** website. See [build_site\(\)](#) for more information.

Value

Paths of files of which the checksums have changed.

read_toml

*Read and write TOML data (Tom's Obvious Markup Language)***Description**

The function `read_toml()` reads TOML data from a file or a character vector, and the function `write_toml()` converts an R object to TOML.

Usage

```
read_toml(file, x = read_utf8(file), strict = TRUE)
```

```
write_toml(x, output = NULL)
```

```
toml2yaml(file, output = NULL)
```

```
yaml2toml(file, output = NULL)
```

Arguments

<code>file</code>	Path to an input (TOML or YAML) file.
<code>x</code>	For <code>read_toml()</code> , the TOML data as a character vector (it is read from <code>file</code> by default; if provided, <code>file</code> will be ignored). For <code>write_toml()</code> , an R object to be converted to TOML.
<code>strict</code>	Whether to try RcppTOML and Hugo only (i.e., not to use the naive parser). If <code>FALSE</code> , only the naive parser is used (this is not recommended, unless you are sure your TOML data is really simple).
<code>output</code>	Path to an output file. If <code>NULL</code> , the TOML data is returned, otherwise the data is written to the specified file.

Details

For `read_toml()`, it first tries to use the R package **RcppTOML** to read the TOML data. If **RcppTOML** is not available, it uses Hugo to convert the TOML data to YAML, and reads the YAML data via the R package **yaml**. If Hugo is not available, it falls back to a naive parser, which is only able to parse top-level fields in the TOML data, and it only supports character, logical, and numeric (including integer) scalars.

For `write_toml()`, it converts an R object to YAML via the R package **yaml**, and uses Hugo to convert the YAML data to TOML.

Value

For `read_toml()`, an R object. For `write_toml()`, `toml2yaml()`, and `yaml2toml()`, a character vector (marked by `xfun::raw_string()`) of the TOML/YAML data if `output = NULL`, otherwise the TOML/YAML data is written to the output file.

Examples

```
## Not run:
v = blogdown::read_toml(x = c("a = 1", "b = true", "c = \"Hello\"", "d = [1, 2]"))
v
blogdown::write_toml(v)

## End(Not run)
```

serve_site

Live preview a site

Description

The function `serve_site()` executes the server command of a static site generator (e.g., hugo server or jekyll server) to start a local web server, which watches for changes in the site, rebuilds the site if necessary, and refreshes the web page automatically; `stop_server()` stops the web server.

Usage

```
serve_site(...)

stop_server()
```

Arguments

... Arguments passed to `servr::server_config()` (only arguments `host`, `port`, `browser`, `daemon`, and `interval` are supported).

Details

By default, the server also watches for changes in R Markdown files, and recompile them automatically if they are modified. This means they will be automatically recompiled once you save them. If you do not like this behavior, you may set `options(blogdown.knit.on_save = FALSE)` (ideally in your `‘.Rprofile’`). When this feature is disabled, you will have to manually compile Rmd documents, e.g., by clicking the Knit button in RStudio.

The site generator is defined by the global R option `blogdown.generator`, with the default being `‘hugo’`. You may use other site generators including `jekyll` and `hexo`, e.g., `options(blogdown.generator = ‘jekyll’)`. You can define command-line arguments to be passed to the server of the site generator via the global R option `blogdown.X.server`, where `X` is `hugo`, `jekyll`, or `hexo`. The default for Hugo is `options(blogdown.hugo.server = c(‘-D’, ‘-F’, ‘--navigateToChanged’))` (see the documentation of Hugo server at https://gohugo.io/commands/hugo_server/ for the meaning of these arguments).

Note

For the Hugo server, the argument `--navigateToChanged` is used by default, which means when you edit and save a source file, Hugo will automatically navigate the web browser to the page corresponding to this source file (if the page exists). However, due to a Hugo bug (<https://github.com/gohugoio/hugo/issues/3811>), this automatic navigation may not always work for R Markdown posts, and you may have to manually refresh your browser. It should work reliably for pure Markdown posts, though.

shortcode

Helper functions to write Hugo shortcodes using the R syntax

Description

These functions return Hugo shortcodes with the shortcode name and arguments you specify. The closing shortcode will be added only if the inner content is not empty. The function `shortcode_html()` is essentially `shortcode(.type = 'html')`. The function `shortcodes()` is a vectorized version of `shortcode()`. The paired functions `shortcode_open()` and `shortcode_close()` provide an alternative method to open and close shortcodes, which allows inner content be processed safely by Pandoc (e.g., citation keys in the content).

Usage

```
shortcode(.name, ..., .content = NULL, .type = "markdown")
```

```
shortcode_html(...)
```

```
shortcodes(..., .sep = "\n")
```

```
shortcode_open(...)
```

```
shortcode_close(...)
```

Arguments

<code>.name</code>	The name of the shortcode.
<code>...</code>	All arguments of the shortcode (either all named, or all unnamed). The ... arguments of all other functions are passed to <code>shortcode()</code> .
<code>.content</code>	The inner content for the shortcode.
<code>.type</code>	The type of the shortcode: <code>markdown</code> or <code>html</code> .
<code>.sep</code>	The separator between two shortcodes (by default, a newline).

Details

These functions can be used in either **knitr** inline R expressions or code chunks. The returned character string is wrapped in `htmltools::HTML()`, so **rmarkdown** will protect it from the Pandoc conversion. You cannot simply write `{{< shortcode >}}` in R Markdown, because Pandoc is not aware of Hugo shortcodes, and may convert special characters so that Hugo can no longer recognize the shortcodes (e.g. `<` will be converted to `<`).

If your document is pure Markdown, you can use the Hugo syntax to write shortcodes, and there is no need to call these R functions.

Value

A character string wrapped in `htmltools::HTML()`; `shortcode()` returns a string of the form `{{% name args %}}`, and `shortcode_html()` returns `{{< name args >}}`.

Note

Since Hugo v0.60, Hugo has switched its default Markdown rendering engine to Goldmark. One consequence is that shortcodes may fail to render. You may enable the unsafe option in the configuration file: <https://gohugo.io/getting-started/configuration-markup/#goldmark>.

References

<https://gohugo.io/extras/shortcodes/>

Examples

```
library(blogdown)

shortcode("tweet", "1234567")
shortcodes("tweet", as.character(1:5)) # multiple tweets
shortcode("figure", src = "/images/foo.png", alt = "A nice figure")
shortcode("highlight", "bash", .content = "echo hello world;")

shortcode_html("myshortcode", .content = "My <strong>shortcode</strong>.")

shortcode_open("figure", src = "/images/foo.png")
# This inner text will be *processed* by Pandoc, @Smith2006
shortcode_close("figure")
```

Index

blogdown, [2](#)
blogdown-package (blogdown), [2](#)
build_dir, [3](#)
build_site, [3](#), [10](#), [13](#)
bundle_site, [4](#)

commandArgs, [4](#)
count_yaml (find_yaml), [6](#)

dep_path, [5](#)

find_categories (find_yaml), [6](#)
find_tags (find_yaml), [6](#)
find_yaml, [6](#)

HTML, [17](#)
html_document2, [7](#)
html_page, [7](#)
hugo_available (hugo_cmd), [8](#)
hugo_build, [3](#)
hugo_build (hugo_cmd), [8](#)
hugo_cmd, [4](#), [8](#)
hugo_convert (hugo_cmd), [8](#)
hugo_server (hugo_cmd), [8](#)
hugo_version (hugo_cmd), [8](#)

identity, [4](#)
install_hugo, [11](#)
install_theme, [12](#)

knit, [4](#)

md5sum_filter, [4](#), [13](#)

new_content (hugo_cmd), [8](#)
new_post (hugo_cmd), [8](#)
new_site, [2](#)
new_site (hugo_cmd), [8](#)

output_format, [7](#)

raw_string, [14](#)

read_toml, [14](#)
render, [3](#), [4](#)

serve_site, [3](#), [4](#), [15](#)
server_config, [10](#), [15](#)
shortcode, [16](#)
shortcode_close (shortcode), [16](#)
shortcode_html (shortcode), [16](#)
shortcode_open (shortcode), [16](#)
shortcodes (shortcode), [16](#)
stop_server (serve_site), [15](#)
system2, [8](#)

timestamp_filter, [4](#)
timestamp_filter (md5sum_filter), [13](#)
toml2yaml (read_toml), [14](#)
toTitleCase, [10](#)

update_hugo (install_hugo), [11](#)

write_toml (read_toml), [14](#)

yaml2toml (read_toml), [14](#)