Package 'blogdown'

January 18, 2026

Type Package

Title Create Blogs and Websites with R Markdown

Version 1.23

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

URL https://github.com/rstudio/blogdown, https://pkgs.rstudio.com/blogdown/

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

Depends R (>= 3.5.0)

Imports bookdown (>= 0.22), htmltools, httpuv (>= 1.4.0), jsonlite, knitr (>= 1.25), later, rmarkdown (>= 2.8), servr (>= 0.21), xfun (>= 0.34), yaml (>= 2.1.19)

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

Config/Needs/website pkgdown, tidyverse/tidytemplate, rstudio/quillt, rstudio/webshot2

Encoding UTF-8

RoxygenNote 7.3.3

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

NeedsCompilation no

Author Yihui Xie [aut, cre] (ORCID: <https://orcid.org/0000-0003-0645-5666>), Christophe Dervieux [aut] (ORCID: <https://orcid.org/0000-0003-4474-2498>), Alison Presmanes Hill [aut] (ORCID: <https://orcid.org/0000-0002-8082-1890>), Amber Thomas [ctb],

Beilei Bian [ctb],
Brandon Greenwell [ctb],
Brian Barkley [ctb],
Deependra Dhakal [ctb],
Eric Nantz [ctb],
Forest Fang [ctb],
Garrick Aden-Buie [ctb],
Hiroaki Yutani [ctb],
Ian Lyttle [ctb],
Jake Barlow [ctb],
James Balamuta [ctb],
JJ Allaire [ctb],
Jon Calder [ctb],
Jozef Hajnala [ctb],
Juan Manuel Vazquez [ctb],
Kevin Ushey [ctb],
Leonardo Collado-Torres [ctb],
Maëlle Salmon [ctb],
Maria Paula Caldas [ctb],
Nicolas Roelandt [ctb],
Oliver Madsen [ctb],
Raniere Silva [ctb],
TC Zhang [ctb],
Xianying Tan [ctb],
Posit Software, PBC [cph, fnd]

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2026-01-18 04:20:02 UTC

Contents

`blogdown` *The* **blogdown** *package*

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` *Build all Rmd files under a directory*

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 *Build a website*

Description

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

Usage

```
build_site(local = FALSE, run_hugo = TRUE, build_rmd = FALSE, ...)
```

Arguments

local	Whether to build the website locally. This argument is passed to hugo_build(), and local = TRUE is mainly for serving the site locally via serve_site().
run_hugo	Whether to run hugo_build() after R Markdown files are compiled.
build_rmd	Whether to (re)build R Markdown files. By default, they are not built. See 'Details' for how build_rmd = TRUE works. Alternatively, it can take a vector of file paths, which means these files are to be (re)built. Or you can provide a function that takes a vector of paths of all R Markdown files under the 'content/' directory, and returns a vector of paths of files to be built, e.g., build_rmd = blogdown::filter_timestamp. A few aliases are currently provided for such functions: build_rmd = 'newfile' is equivalent to build_rmd = blogdown::filter_newfile, build_rmd = 'timestamp' is equivalent to build_rmd = blogdown::filter_timestamp, and build_rmd = 'md5sum' is equivalent to build_rmd = blogdown::filter_md5sum.
...	Other arguments to be passed to hugo_build().

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 R Markdown posts, there are a few possible rendering methods: html (the default), markdown, and custom. The method can be set in the global option blogdown.method (usually in the '.Rprofile' file), e.g., options(blogdown.method = "custom").

For the html method, '.Rmd' posts are rendered to '.html' via rmarkdown::render(), which means Markdown is processed through Pandoc. For the markdown method, '.Rmd' is rendered to '.md', which will typically be rendered to HTML later by the site generator such as Hugo.

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()). The custom method 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 other rendering methods, 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::filter_newfile`, which means to build new Rmd files that have not been built before, or `blogdown::filter_timestamp` to build Rmd files if their time stamps (modification time) are newer than their output files, or `blogdown::filter_md5sum`, 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

`dir`	The root directory of the website project (should contain a 'content/' folder).
`output`	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 `bundle_site()`. You should modify the website in place *only if you have a backup for this directory or it is under version control*.

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)
```

`check_site` *Provide diagnostics for a website project*

Description

The function `check_site()` runs all `check_*()` functions on this page against a website project. See 'Details' for what each `check_*()` function does.

Usage

```
check_site()

check_config()

check_gitignore()

check_hugo()

check_netlify()

check_vercel()

check_content()
```

Details

`check_config()` checks the configuration file ('`config.yaml`' or '`config.toml`') for settings such as `baseURL` and `ignoreFiles`.

`check_gitignore()` checks if necessary files are incorrectly ignored in GIT.

`check_hugo()` checks possible problems with the Hugo installation and version.

`check_netlify()` checks the Hugo version specification and the publish directory in the Netlify config file '`netlify.toml`'. Specifically, it will check if the local Hugo version matches the version specified in '`netlify.toml`' (in the environment variable *HUGO_VERSION*), and if the *publish* setting in '`netlify.toml`' matches the *publishDir* setting in Hugo's config file (if it is set).

`check_vercel()` checks if the Hugo version specified in '`vercel.json`' (if it exists) matches the Hugo version used in the current system.

`check_content()` checks for possible problems in the content files. First, it checks for the validity of YAML metadata of all posts. Then it searches for posts with future dates and draft posts, and

lists them if found (such posts appear in the local preview by default, but will be ignored by default when building the site). Then it checks for R Markdown posts that have not been rendered, or have output files older than the source files, and plain Markdown posts that have '.html' output files (which they should not have). At last, it detects '.html' files that seem to be generated by clicking the Knit button in RStudio with **blogdown** < v0.21. Such '.html' files should be deleted, since the Knit button only works with **blogdown** >= v0.21.

`clean_duplicates` *Clean duplicated output files*

Description

For an output file 'FOO.html', 'FOO.md' should be deleted if 'FOO.Rmd' exists, and 'FOO.html' should be deleted when 'FOO.Rmarkdown' exists (because 'FOO.Rmarkdown' should generate 'FOO.markdown' instead) or neither 'FOO.Rmarkdown' nor 'FOO.Rmd' exists (because a plain Markdown file should not be knitted to HTML).

Usage

```
clean_duplicates(preview = TRUE)
```

Arguments

`preview`	Whether to preview the file list, or just delete the files. If you are sure the files can be safely deleted, use `preview = FALSE`.

Value

For `preview = TRUE`, a logical vector indicating if each file was successfully deleted; for `preview = FALSE`, the file list is printed.

`config_netlify` *Create the configuration (file) for Netlify*

Description

This function provides some default configurations for a Huge website to be built via Hugo and deployed on Netlify. It sets the build command for the production and preview contexts, respectively (for preview contexts such as 'deploy-preview', the command will build future posts). It also sets the publish directory according to your setting in Hugo's config file (if it exists, otherwise it will be the default 'public' directory). The Hugo version is set to the current version of Hugo found on your computer.

Usage

```
config_netlify(output = "netlify.toml", new_config = list())
```

Arguments

output	Path to the output file, or NULL. If the file exists and the R session is interactive, you will be prompted to decide whether to overwrite the file.
new_config	If any default configuration does not apply to your site, you may provide a list of configurations to override the default. For example, if you want to use Hugo v0.25.1, you may use new_config = list(build = list(environment = list(HUGO_VERSION = '0.25.1'))).

Value

If output = NULL, a character vector of TOML data representing the configurations (which you can preview and decide whether to write it to a file), otherwise the TOML data is written to a file.

References

See Netlify's documentation on the configuration file 'netlify.toml' for the possible settings: https://docs.netlify.com/configure-builds/file-based-configuration/

Examples

```
blogdown::config_netlify(output = NULL)  # default data

# change the publish dir to 'docs/'
blogdown::config_netlify(NULL, list(build = list(publish = "docs")))
```

---

config_Rprofile *Create or modify the '.Rprofile' file for a website project*

Description

If the file '.Rprofile' does not exist in the current directory, copy the file from the 'resources' directory of **blogdown**. If the option blogdown.hugo.version is not found in this file, append options(blogdown.hugo.version = "VERSION") to it, where VERSION is obtained from hugo_version().

Usage

```
config_Rprofile()
```

Value

As a side-effect, the file '.Rprofile' is created or modified.

config_vercel *Create the configuration file for Vercel*

Description

Create 'vercel.json' that contains the Hugo version currently used.

Usage

```
config_vercel(output = "vercel.json")
```

Arguments

output	Path to the output file, or NULL to print the config.

References

Vercel: https://vercel.com

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 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.

`filter_newfile` *Look for files that have been possibly modified or out-of-date*

Description

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

Usage

```
filter_newfile(files)

filter_timestamp(files)

filter_md5sum(files, db = "blogdown/md5sum.txt")
```

Arguments

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

Details

The function `filter_newfile()` returns paths of source files that do not have corresponding output files, e.g., an '`.Rmd`' file that doesn't have the '`.html`' output file.

The function `filter_timestamp()` compares the modification time of an Rmd file with that of its output file, and returns the path of a file if it is 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.

The function `filter_md5sum()` 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.

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

Value

The filtered file paths.

find_hugo *Find or remove the Hugo executable*

Description

Search for Hugo in a series of possible installation directories (see `install_hugo()` for these directories) with `find_hugo()`, or remove the Hugo executable(s) found with `remove_hugo()`.

Usage

```
find_hugo(version = getOption("blogdown.hugo.version"), quiet = FALSE)

remove_hugo(version = getOption("blogdown.hugo.version"), force = FALSE)
```

Arguments

`version`	The expected version number, e.g., `'0.25.1'`. If `NULL`, it will try to find/remove the maximum possible version. If `'all'`, find/remove all possible versions. In an interactive R session when `version` is not provided, `remove_hugo()` will list all installed versions of Hugo, and you can select which versions to remove.
`quiet`	Whether to signal a message when two versions of Hugo are found: one is found on the system *PATH* variable, and one is installed by `install_hugo()`.
`force`	By default, `remove_hugo()` only removes Hugo installed via `install_hugo()`. For `force = TRUE`, it will try to remove any Hugo executables found via `find_hugo()`.

Details

If your website depends on a specific version of Hugo, we strongly recommend that you set `options(blogdown.hugo.versio = )` to the version number you desire in the file `.Rprofile` in the root directory of the website project, so that **blogdown** can try to find the right version of Hugo before it builds or serves the website. You can use the function `config_Rprofile()` to do this automatically.

Value

For `find_hugo()`, it returns the path to the Hugo executable if found, otherwise it will signal an error, with a hint on how to install (the required version of) Hugo. If Hugo is found via the environment variable *PATH*, only the base name of the path is returned (you may use `Sys.which('hugo')` to obtain the full path).

If `version = 'all'`, return the paths of all versions of Hugo installed.

`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

`field`, `fields`	A character vector of YAML field names.
`value`	A vector of the field values to be matched.
`open`	Whether to open the matched files automatically.
`sort_by_count`	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)
```

html_page *An R Markdown output format for* **blogdown** *web pages*

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,
  pandoc_args = c("-M", "link-citations=true", "--preserve-tabs"),
  keep_md = FALSE,
  pre_knit = NULL,
  post_processor = NULL
)
```

Arguments

`...`, `number_sections`, `self_contained`, `highlight`, `template`, `pandoc_args`
: 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", exact = FALSE)

hugo_build(
  local = FALSE,
  args = getOption("blogdown.hugo.args"),
  baseURL = NULL,
  relativeURLs = NULL
)

new_site(
  dir = ".",
  force = NA,
  install_hugo = TRUE,
  format = "yaml",
  sample = TRUE,
  theme = "yihui/hugo-lithium",
  hostname = "github.com",
  theme_example = TRUE,
  empty_dirs = FALSE,
  to_yaml = TRUE,
  netlify = TRUE,
  .Rprofile = TRUE,
  serve = if (interactive()) "ask" else FALSE
)

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

new_post(
  title,
```

```
  kind = "",
  open = interactive(),
  author = getOption("blogdown.author"),
  categories = NULL,
  tags = NULL,
  date = Sys.Date(),
  time = getOption("blogdown.time", FALSE),
  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 `system2('hugo', ...)`, e.g. `new_content(path)` is basically `hugo_cmd(c('new', path))` (i.e. run the command `hugo new path`).
`version`	A version number.
`exact`	If `FALSE`, check if the current Hugo version is equal to or higher than the specified `version`. If `TRUE`, check if the exact version is available.
`local`	Whether to build the site for local preview (if `TRUE`, all drafts and future posts will also be built).
`args`	A character vector of command-line arguments to be passed to hugo, e.g., `c("--minify", "--quiet")`.
`baseURL, relativeURLs`	Custom values of `baseURL` and `relativeURLs` to override Hugo's default and the settings in the site's config file.
`dir`	The directory of the new site.
`force`	Whether to create the site in a directory even if it is not empty. By default, `force = TRUE` when the directory only contains hidden, RStudio project ('`*.Rproj`'), '`LICENSE`', and/or '`README`' files.
`install_hugo`	Whether to install Hugo automatically if it is not found.
`format`	The format of the configuration file, e.g., `'yaml'` or `'toml'` (the value `TRUE` will be treated as `'yaml'`, and `FALSE` means `'toml'`). Note that the frontmatter of the new (R) Markdown file created by `new_content()` always uses YAML instead of TOML or JSON.
`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 `user/repo`, and you can optionally specify a GIT branch or tag name after `@`, i.e. `theme` can be of the form `user/repo@branch`). You can also specify a full URL to the zip file or

	tarball of the theme. If `theme = NA`, no themes will be installed, and you have to manually install a theme.
`hostname`	Where to find the theme. Defaults to `github.com`; 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 '`exampleSite`' 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.
`netlify`	Whether to create a Netlify config file '`netlify.toml`'.
`.Rprofile`	Whether to create a '`.Rprofile`' file. If `TRUE`, a sample '`.Rprofile`' will be created. It contains some global options, such as `options(blogdown.hugo.version)`, which makes sure you will use a specific version of Hugo for this site in the future.
`serve`	Whether to start a local server to serve the site. By default, this function will ask you in an interactive R session if you want to serve the site.
`path`	The path to the new file under the '`content`' 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., `post/`.
`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.
`time`	Whether to include the time of the day in the `date` field of the post. If `TRUE`, the `date` will be of the format '`%Y-%m-%dT%H:%M:%S%z`' (e.g., '`2001-02-03T04:05:06-0700`'). Alternatively, it can take a character string to be appended to the `date`. It can be important and helpful to include the time in the date of a post. For example, if your website is built on a server (such as Netlify or Vercel) and your local timezone is ahead of UTC, your local date may be a *future* date on the server, and Hugo will not build future posts by default (unless you use the `-F` flag).
`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/2020-07-23-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 `tools::toTitleCase()`). 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 servr::server_config().

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 of a certain version (or above if exact = FALSE) 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()
```

hugo_installers *Available Hugo installers of a version*

Description

Given a version number, return the information of available installers. If `install_hugo()` fails, you may run this function to check the available installers and obtain their `os`/`arch` info.

Usage

```
hugo_installers(version = "latest")
```

Arguments

`version`	A version number. The default is to automatically detect the latest version. Versions before v0.17 are not supported.

Value

A data frame containing columns `os` (operating system), `arch` (architecture), `extended` (extended version or not), `withdeploy` (with the deploy feature or not). If your R version is lower than 4.1.0, a character vector of the installer filenames will be returned instead.

Examples

```
blogdown::hugo_installers()
blogdown::hugo_installers("0.89.0")
blogdown::hugo_installers("0.17")
```

install_hugo *Install Hugo*

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.

Usage

```
install_hugo(
  version = "latest",
  extended = TRUE,
  arch = "auto",
  os = "auto",
  force = FALSE,
```

```
  ...
)

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.
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. Note that this feature is not available to Hugo version lower than v0.43. It also requires a 64-bit system; if the system is based on the ARM architecture, only macOS is supported at the moment.
arch, os	The architecture and operating system name. These arguments, along with version and extended, determines the filename of the Hugo installer. See https://github.com/gohugoio/hugo/releases for all of Hugo's installers. By default, the argument values are automatically detected. In case the detection should fail, you can provide the values manually, e.g., extended = FALSE, arch = 'ARM64', and os = 'FreeBSD' would install 'hugo_*_FreeBSD-ARM.tar.gz'.
force	Whether to reinstall Hugo if the specified version has been installed.
...	Ignored.

Details

This function tries to install Hugo to Sys.getenv('APPDATA') on Windows, '~/Library/Application Support' on macOS, and '~/.local/share' on other platforms (such as Linux). The hugo executable is installed to a subdirectory with the Hugo version number being its name, e.g., '~/Library/Application Support/Hugo/0.76.5'. If these directories are not writable, the R 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 macOS 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'). 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.

Note

For macOS users, you are not recommended to install Hugo via Homebrew, because you may accidentally update it to the latest version, which might break your existing sites.

See Also

remove_hugo() to remove Hugo.

---

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 user/repo, and you can optionally specify a GIT branch or tag name after @, i.e. theme can be of the form user/repo@branch). You can also specify a full URL to the zip file or tarball of the theme. If theme = NA, no themes will be installed, and you have to manually install a theme.
hostname	Where to find the theme. Defaults to github.com; 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 '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.

`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

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

stop_server()
```

Arguments

`...`	Arguments passed to `servr::server_config()` (only arguments `host`, `port`, `browser`, `daemon`, and `interval` are supported).
`.site_dir`	Directory to search for site configuration file. It defaults to `getwd()`, and can also be specified via the global option `blogdown.site_root`.

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).

`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

`.name`	The name of the shortcode.
`...`	All arguments of the shortcode (either all named, or all unnamed). The `...` arguments of all other functions are passed to `shortcode()`.
`.content`	The inner content for the shortcode.
`.type`	The type of the shortcode: `markdown` or `html`.
`.sep`	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", user = "SanDiegoZoo", id = "1453110110599868418")
# multiple tweets (id's are fake)
shortcodes("tweet", user = "SanDiegoZoo", id = as.character(1:5))
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