

# Package ‘servr’

October 19, 2020

**Type** Package

**Title** A Simple HTTP Server to Serve Static Files or Dynamic Documents

**Version** 0.20

**Maintainer** Yihui Xie <xie@yihui.name>

**Description** Start an HTTP server in R to serve static files, or dynamic documents that can be converted to HTML files (e.g., R Markdown) under a given directory.

**Depends** R (>= 3.0.0)

**Imports** mime (>= 0.2), httpuv (>= 1.5.2), xfun, jsonlite

**Suggests** tools, later, rstudioapi, knitr (>= 1.9), rmarkdown

**License** GPL

**URL** <https://github.com/yihui/servr>

**BugReports** <https://github.com/yihui/servr/issues>

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Yihui Xie [aut, cre] (<<https://orcid.org/0000-0003-0645-5666>>),  
Carson Sievert [ctb],  
Jesse Anderson [ctb],  
Ramnath Vaidyanathan [ctb]

**Repository** CRAN

**Date/Publication** 2020-10-19 04:40:02 UTC

## R topics documented:

browse_last	2
daemon_stop	2
httd	3
jeekyll	4
make	6

random_port . . . . .	7
server_config . . . . .	7
serve_example . . . . .	9
vign . . . . .	9

<b>Index</b>	<b>11</b>
--------------	-----------

---

browse_last	<i>Reopen the last browsed page</i>
-------------	-------------------------------------

---

### Description

If you have launched a page in the browser via `servr` but closed it later, you may call this function to reopen it.

### Usage

```
browse_last(open = TRUE)
```

### Arguments

open	Whether to reopen the lastly browsed page. If FALSE, the URL of the previously browsed page will be returned.
------	---------------------------------------------------------------------------------------------------------------

### Examples

```
servr::browse_last()
```

---

daemon_stop	<i>Utilities for daemonized servers</i>
-------------	-----------------------------------------

---

### Description

`daemon_list()` returns IDs of servers, which can be used to stop the daemonized servers.

### Usage

```
daemon_stop(which = daemon_list())
```

```
daemon_list()
```

### Arguments

which	A integer vector of the server IDs; by default, IDs of all existing servers in the current R session obtained from <code>daemon_list()</code> , i.e., all daemon servers will be stopped by default.
-------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Value**

The function `daemon_list()` returns a list of existing server IDs, and `daemon_stop()` returns an invisible `NULL`.

---

<code>httd</code>	<i>Serve static files under a directory</i>
-------------------	---------------------------------------------

---

**Description**

If there is an `'index.html'` under this directory, it will be displayed; otherwise the list of files is displayed, with links on their names. After we run this function, we can go to `'http://localhost:port'` to browse the web pages either created from R or read from HTML files.

**Usage**

```
httd(dir = ".", ...)
```

```
httw(
  dir = ".",
  watch = ".",
  pattern = NULL,
  all_files = FALSE,
  handler = NULL,
  ...
)
```

**Arguments**

<code>dir</code>	The root directory to serve.
<code>...</code>	server configurations passed to <code>server_config()</code>
<code>watch</code>	a directory under which <code>httw()</code> is to watch for changes; if it is a relative path, it is relative to the <code>dir</code> argument
<code>pattern</code>	a regular expression passed to <code>list.files()</code> to determine the files to watch
<code>all_files</code>	whether to watch all files including the hidden files
<code>handler</code>	a function to be called every time any files are changed or added under the directory; its argument is a character vector of the filenames of the files modified or added

**Details**

`httd()` is a pure static server, and `httw()` is similar but watches for changes under the directory: if an HTML file is being viewed in the browser, and any files are modified under the directory, the HTML page will be automatically refreshed.

## References

<https://github.com/yihui/srvr>

## Examples

```
#' see https://github.com/yihui/srvr for command line usage
# or run inside an R session
if (interactive()) srvr::httd()
```

---

jekyll

*Serve R Markdown based websites*

---

## Description

R Markdown documents (with the filename extension ‘.Rmd’) are re-compiled using **knitr** or **rmarkdown** when necessary (source files are newer than output files), and the HTML pages will be automatically refreshed in the web browser accordingly.

## Usage

```
jekyll(
  dir = ".",
  input = c(".", "_source", "_posts"),
  output = c(".", "_posts", "_posts"),
  script = c("Makefile", "build.R"),
  serve = TRUE,
  command = "jekyll build",
  ...
)

rmdv2(dir = ".", script = c("Makefile", "build.R"), in_session = FALSE, ...)

rmdv1(dir = ".", script = c("Makefile", "build.R"), in_session = FALSE, ...)
```

## Arguments

dir	the root directory of the website
input	the input directories that contain R Markdown documents (the directories must be relative instead of absolute; same for output directories)
output	the output directories corresponding to input; for an input document ‘foo.Rmd’ under the directory input[i], its output document ‘foo.md’ (or ‘foo.html’) is generated under output[i] if the output document is older than the input document

script	a Makefile (see <a href="#">make</a> ), or (if Makefile not found) the name of an R script to re-build R Markdown documents, which will be executed via command line of the form <code>Rscript build.R arg1 arg2</code> where <code>build.R</code> is the script specified by this argument, <code>arg1</code> is the input filename, and <code>arg2</code> is the output filename; inside the R script, you can use <a href="#">commandArgs(TRUE)</a> to capture <code>c(arg1, arg2)</code> , e.g. <code>knitr::knit(commandArgs(TRUE)[1], commandArgs(TRUE)[2])</code> ; if this R script is not found, either, internal compiling methods will be used, which are basically <a href="#">knit()</a> , <a href="#">knit2html()</a> , or <a href="#">render()</a>
serve	whether to serve the website; if <code>FALSE</code> , the R Markdown documents and the website will be compiled but not served
command	a command to build the Jekyll website; by default, it is <code>jekyll build</code> , and you can use alternative commands, such as <code>bundle exec jekyll build</code>
...	server configurations passed to <a href="#">server_config()</a>
in_session	whether to render the R Markdown documents in the current R session ( <code>TRUE</code> ) or in a separate new R session ( <code>FALSE</code> ); if the former, the argument <code>script</code> can be a function with two arguments, the filenames of the source document and the output document, respectively; an internal function (basically <code>rmarkdown::render()</code> or <code>knitr::knit2html()</code> ) will be used if the <code>script</code> argument is not a function and <code>in_session = TRUE</code>

## Details

The function `jekyll()` sets up a web server to serve a Jekyll-based website. A connection is established between R and the HTML pages through WebSockets so that R can notify the HTML pages to refresh themselves if any R Markdown documents have been re-compiled.

The functions `rmdv1()` and `rmdv2()` are similar to `jekyll()`, and the only difference is the way to compile R Markdown documents: `rmdv1()` uses the **markdown** package (a.k.a R Markdown v1) via [knit2html\(\)](#), and `rmdv2()` calls [render\(\)](#) in the **rmarkdown** package (a.k.a R Markdown v2).

## Note

Apparently `jekyll()` and `rmdv1()` require the **knitr** package, and `rmdv2()` requires **rmarkdown**. You have to install them before calling the server functions here.

All R Markdown documents are compiled in separate R sessions by default. If you have any R Markdown documents that should not be compiled as standalone documents (e.g. child documents), you can use different filename extensions, such as `‘.Rmarkdown’`.

The `baseurl` argument does not work in `jekyll()`, and the base URL setting will be read from `‘_config.yml’` (the `‘baseurl’` field) of the website if present. You should not pass `baseurl` to the function `jekyll()` directly.

For the sake of reproducibility, you are recommended to compile each source document in a separate R session (i.e., use the default `in_session = FALSE`) to make sure they can compile on their own, otherwise the current workspace may affect the evaluation of the code chunks in these source documents. Sometimes it might be useful to compile a document in the current R session. For example, if reading data is time-consuming and it is not convenient to cache it (using the **knitr** chunk option `cache = TRUE`), you may read the data once, temporarily turn off the evaluation of that code

chunk, and keep on working on the rest of code chunks so that data will not be read over and over again.

## References

R Markdown v1: <https://cran.r-project.org/package=markdown>. R Markdown v2: <https://rmarkdown.rstudio.com>. For Jekyll, see <https://jekyllrb.com>. The GitHub repository <https://github.com/yihui/blogdown-jekyll> is an example of serving Jekyll websites with `servr::jekyll()`.

## See Also

The **blogdown** package (based on Hugo and R Markdown v2) is a better alternative to Jekyll: <https://github.com/rstudio/blogdown/>. I strongly recommend you to try it.

## Examples

```
if (interactive()) servr::rmdv1() # serve the current dir with R Markdown v1
if (interactive()) servr::rmdv2() # or R Markdown v2

# built-in examples
servr::serve_example("rmd", servr::rmdv1)
servr::serve_example("rmd", servr::rmdv2)
```

---

make

*Serve files under a directory based on GNU Make*

---

## Description

You can define how and when to rebuild files (such as R Markdown files) using Make rules, e.g. a rule `_posts/%.md: _source/%.Rmd` with a command to build `‘.Rmd’` to `‘.md’` will be executed if and only if `‘foo.Rmd’` is newer than `‘foo.md’`. The exit status of the command `make -q` will decide whether to rebuild files: rebuilding occurs only when the exit code is not 0. When an HTML file has been rebuilt, it will be automatically refreshed in the web browser.

## Usage

```
make(dir = ".", ...)
```

## Arguments

<code>dir</code>	The root directory to serve.
<code>...</code>	server configurations passed to <code>server_config()</code>

## Note

You must have installed GNU Make to use this function. This is normally not a problem for Linux and OS X users (it should be available by default). For Windows users, you can either install GNU Make, or just install **Rtools**, which also contains GNU Make.

## Examples

```
# some built-in examples (if you are not familiar with make, you can take a look
# at the Makefile of each example)
servr::serve_example("make1", servr::make)
servr::serve_example("make2", servr::make)
```

---

random_port	<i>Find a random available TCP port</i>
-------------	-----------------------------------------

---

## Description

Test a series of random TCP ports from 3000 to 8000 (excluding a few that are considered unsafe by Chrome) and return the first available one. A web server can be later started on this port.

## Usage

```
random_port(port = 4321L, host = "127.0.0.1", n = 20)
```

## Arguments

port	The preferred port(s).
host	A string that is a valid IPv4 address that is owned by this server, or "0.0.0.0" to listen on all IP addresses.
n	The maximum number of random ports to be tested.

## Value

A port number, or an error if no ports are available.

---

server_config	<i>Server configurations</i>
---------------	------------------------------

---

## Description

The server functions in this package are configured through this function.

**Usage**

```
server_config(
  dir = ".",
  host = getOption("servr.host", "127.0.0.1"),
  port,
  browser,
  daemon,
  interval = getOption("servr.interval", 1),
  baseurl = "",
  initpath = "",
  verbose = TRUE
)
```

**Arguments**

dir	The root directory to serve.
host	A string that is a valid IPv4 address that is owned by this server, or "0.0.0.0" to listen on all IP addresses.
port	The TCP port number. If it is not explicitly set, the default value will be looked up in this order: First, the command line argument of the form -pNNNN (N is a digit from 0 to 9). If it was passed to R when R was started, NNNN will be used as the port number. Second, the environment variable R_SERVR_PORT. Third, the global option <code>servr.port</code> (e.g., <code>options(servr.port = 4322)</code> ). If none of these command-line arguments, variables, or options were set, the default port will be 4321. If this port is not available, a random available port will be used.
browser	Whether to launch the default web browser. By default, it is TRUE if the R session is <code>interactive()</code> , or when a command line argument -b was passed to R (see <code>commandArgs()</code> ). N.B. the RStudio viewer is used as the web browser if available.
daemon	Whether to launch a daemonized server (the server does not block the current R session) or a blocking server. By default, it is the global option <code>getOption('servr.daemon')</code> (e.g., you can set <code>options(servr.daemon = TRUE)</code> ); if this option was not set, <code>daemon = TRUE</code> if a command line argument -d was passed to R (through <code>Rscript</code> ), or the server is running in an interactive R session.
interval	The time interval used to check if an HTML page needs to be rebuilt (by default, it is checked every second).
baseurl	The base URL (the full URL will be <code>http://host:port/baseurl</code> ).
initpath	The initial path in the URL (e.g. you can open a specific HTML file initially).
verbose	Whether to print messages when launching the server.

**Value**

A list of configuration information of the form `list(host, port, start_server = function(app) {}, ...)`.



---

serve_example	<i>A convenience function to serve examples in this package</i>
---------------	-----------------------------------------------------------------

---

**Description**

Use server functions to serve built-in examples of this package.

**Usage**

```
serve_example(name, FUN, ..., run = interactive())
```

**Arguments**

name	the directory name of the example under the directory system. <code>file('examples', package = 'servr')</code>
FUN	a server function that takes the example path as its first argument, e.g. <a href="#">httd</a> , or <a href="#">rmdv1</a>
...	other arguments passed to FUN
run	whether to run the example (this is mainly for R CMD check purposes: the examples will not be really served when the R session is not interactive, so they will not block R CMD check)

**Value**

NULL if `run = FALSE`, otherwise the value returned from `FUN()`.

**Examples**

```
# R Markdown v1 or v2
servr::serve_example("rmd", servr::rmdv1)
servr::serve_example("rmd", servr::rmdv2)

# GNU Make
servr::serve_example("make1", servr::make)
servr::serve_example("make2", servr::make)
```

---

vign	<i>Serve R Markdown/HTML package vignettes</i>
------	------------------------------------------------

---

**Description**

Serve package vignettes under the `'vignettes/'` directory. Because the HTML output files should not be included in the source package, this function renders R Markdown/HTML vignettes, displays them in the web browser, and deletes the HTML output files. You will see the HTML output when you click the links on the `'.Rmd'` or `'.Rhtml'` files (unlike the static HTTP server, the compiled output instead of the source document is displayed).

**Usage**

```
vign(dir = "vignettes", ...)
```

**Arguments**

<code>dir</code>	The root directory to serve.
<code>...</code>	server configurations passed to <code>server_config()</code>

**Details**

When developing R packages, you may want to preview your vignettes once in a while. You can certainly click the button in RStudio to do it, but that requires you to install the package and rebuild the vignettes. With this function, your vignette will be rebuilt automatically when you update the source document. Moreover, because the compilation takes place in the current R session, you can take advantage of `devtools::load_all()` (which has a keyboard shortcut in the RStudio IDE) to reload your package and see the updated vignette in the web browser.

**Note**

You are supposed to call this function from the root directory of your package. If that is not the case, you should provide the correct path to the ‘vignettes/’ directory of your package to the `dir` argument.

# Index

`browse_last`, 2

`commandArgs`, 5, 8

`daemon_list (daemon_stop)`, 2  
`daemon_stop`, 2

`httd`, 3, 9  
`httw (httd)`, 3

`interactive`, 8

`jekyll`, 4

`knit`, 5  
`knit2html`, 5

`list.files`, 3

`make`, 5, 6

`random_port`, 7  
`render`, 5  
`rmdv1`, 9  
`rmdv1 (jekyll)`, 4  
`rmdv2 (jekyll)`, 4

`serve_example`, 9  
`server_config`, 3, 5, 6, 7, 10

`vign`, 9