Options

Chunk options and package options

2020-06-30

The **knitr** package provides a lot of chunk options for customizing nearly all components of code chunks, such as the source code, text output, plots, and the language of the chunk. It also offers some options at the package level to customize the knitting process. This page documents all chunk options and package options available in **knitr**. The default values of these options are in parentheses in the list items.

Chunk Options

Chunk options are written in chunk headers. The syntax for chunk headers depends on the document format, e.g., for `.Rnw` documents (R + LaTeX), chunk headers are written with `<< >>=`, and for `.Rmd` documents, chunk headers are written with ` ```{r} `. The examples below are primarily for `.Rmd` documents (R Markdown), but in most cases, the chunk options can be used with any document format.

Chunk options are written in the form `tag=value` like this:

~~~
```{r, my-chunk, echo=FALSE, fig.height=4, dev='jpeg'}
```
~~~

A special chunk option is the chunk label (e.g., `my-chunk` in the above example). Only the chunk label does not need a `tag` (i.e., you only provide the `value`). If you prefer the form `tag=value`, you could also use the chunk option `label` explicitly, e.g.,

~~~
```{r, label='my-chunk'}
```
~~~

The chunk label for each chunk is assumed to be unique within the document. This is especially important for cache and plot filenames, because these filenames are based on chunk labels.

Chunks without labels will be assigned labels like `unnamed-chunk-i`, where `i` is an incremental number.

You may use `knitr::opts_chunk$set()` to change the default values of chunk options in a document. For example, you may put this in the first code chunk of your document:

~~~
```{r, setup, include=FALSE}
knitr::opts_chunk$set(
  comment = '', fig.width = 6, fig.height = 6
)
```
~~~

Below are a few more tips about chunk options:

1. The chunk header must be written on one line. You must not break the line.

2. Try to avoid spaces, periods (`.`), and underscores (`_`) in chunk labels and paths. If you need separators, you are recommended to use hyphens (`-`) instead. For example, `setup-options` is a good label, whereas `setup.options` and `chunk 1` are bad; `fig.path = 'figures/mcmc-'` is a good path for figure output, and `fig.path = 'markov chain/monte carlo'` is bad.

3. All option values must be *valid R expressions*. You may think of them as values to be passed to function arguments.

   - For example, options that take *character* values must be quoted, e.g., `results = 'asis'` and `out.width = '\\textwidth'` (remember that a literal backslash needs double backslashes).
   - In theory, the chunk label should be quoted, too. However, for the sake of convenience, it will be automatically quoted if you did not (e.g., ` ```{r, 2a} ` will parsed as ` ```{r, '2a'} `).
   - You can write arbitrarily complicated expressions as long as they are valid R code.

Alternatively, you can write chunk options in the body of a code chunk after `#|`, e.g.,

~~~
```{r}
#| my-chunk, echo = FALSE, fig.width = 10,
#| fig.cap = "This is a long long
#|   long long caption."

plot(cars)
```
~~~

For this syntax, chunk options must be written on *continuous* lines (i.e., all lines must start with the special comment prefix such as `#|`) at the beginning of the chunk body. The blank line between the options and code is optional. This syntax allows for hard-wrapping the options. You can break the options onto as many lines as you wish. If the same option is provided in both the chunk body and in the chunk header (inside ```` ```{} ````), the former will override the latter. You can also use the YAML syntax to write options inside a chunk in the form `tag: value`. Normally you have to provide only one option per line, e.g.,

~~~
```{r}
#| echo: false
#| fig.width: 10
```
~~~

If you choose to use the YAML syntax, the option values must be valid YAML values instead of raw R expressions.

Below is a list of chunk options in **knitr** documented in the format "`option`: (`default value`; type of value)".

Code evaluation

- `eval`: (`TRUE`; logical or numeric) Whether to evaluate the code chunk. It can also be a numeric vector to choose which R expression(s) to evaluate, e.g., `eval = c(1, 3, 4)` will evaluate the first, third, and fourth expressions, and `eval = -(4:5)` will evaluate all expressions except the fourth and fifth.

Text output

- `echo`: (`TRUE`; logical or numeric) Whether to display the source code in the output document. Besides `TRUE`/`FALSE`, which shows/hides the source code, we can also use a numeric vector to choose which R expression(s) to echo in a chunk, e.g., `echo = 2:3` means to echo only the 2nd and 3rd expressions, and `echo = -4` means to exclude the 4th expression.

- `results`: (`'markup'`; character) Controls how to display the text results. Note that this option only applies to normal text output (not warnings, messages, or errors). The possible values are as follows:

    - `markup`: Mark up text output with the appropriate environments depending on the output format. For example, for R Markdown, if the text output is a character string `"[1] 1 2 3"`, the actual output that **knitr** produces will be:

~~~
```
[1] 1 2 3
```
~~~

In this case, `results='markup'` means to put the text output in fenced code blocks (` ``` `).

- `asis`: Write text output as-is, i.e., write the raw text results directly into the output document without any markups.

  ~~~
  ```{r, results='asis'}
  cat("I'm raw **Markdown** content.\n")
  ```
  ~~~

- `hold`: Hold all pieces of text output in a chunk and flush them to the end of the chunk.

- `hide` (or `FALSE`): Hide text output.

- `collapse`: (`FALSE`; logical) Whether to, if possible, collapse all the source and output blocks from one code chunk into a single block (by default, they are written to separate blocks). This option only applies to Markdown documents.

- `warning`: (`TRUE`; logical) Whether to preserve warnings (produced by `warning()`) in the output.

  - If `FALSE`, all warnings will be suppressed.

  - If `NA`, warnings will not be captured and will be printed to the console by default.

  - This option can also take numeric values as indices to select a subset of warnings to include in the output. Note that these values reference the indices of the warnings themselves (e.g., `3` means "the third warning thrown from this chunk") and not the indices of which expressions are allowed to emit warnings.

- `message`: (`TRUE`; logical) Whether to preserve messages emitted by `message()` (similar to the option `warning`).

- `error`: (`TRUE`; logical) Whether to preserve errors (from `stop()`). By default, the code evaluation will not stop even in case of errors! If we want to stop on errors, we need to set this option to `FALSE`. Note that R Markdown has changed this default value to `FALSE`. When the chunk option `include = FALSE`, **knitr** will stop on error, because it is easy to overlook potential errors in this case. If you understand this caveat and want to handle potential errors by yourself, you may set `error` to numerical values as they are defined by `evaluate::evaluate()`:

  - `0` will keep evaluating after errors as if you had pasted the text into a terminal;
  - `1` will stop evaluation after an error but then ends normally (to manually handle errors, you can use the `error` hook);
  - `2` will signal errors normally (i.e., it will halt R).

- `include`: (`TRUE`; logical) Whether to include the chunk output in the output document. If `FALSE`, nothing will be written into the output document, but the code is still evaluated and plot files are generated if there are any plots in the chunk, so you can manually insert figures later.

- `tab.cap`: (`NULL`; character) A caption for `kable()` in the code chunk. For this chunk option to work, one code chunk must have only a single `kable()` call.

- `strip.white`: (`TRUE`; logical) Whether to remove blank lines in the beginning or end of a source code block in the output.

- `class.output`: (`NULL`; character) A vector of class names to be added to the text output blocks. This option only works for HTML output formats in R Markdown. For example, with `class.output = c('foo', 'bar')`, the text output will be placed in `<pre class="foo bar"></pre>`.

- `class.message`/`class.warning`/`class.error`: (`NULL`; character) Similar to `class.output`, but applied to messages, warnings, and errors in R Markdown output. Please see the "Code Decoration" section for `class.source`, which applies similarly to source code blocks.

- `attr.output`/`attr.message`/`attr.warning`/`attr.error`: (`NULL`; character) Similar to the `class.*` options above, but for specifying arbitrary fenced code block attributes for Pandoc; `class.*` is a special application of `attr.*`, e.g., `class.source = 'numberLines'` is equivalent to `attr.source = '.numberLines'`, but `attr.source` can take arbitrary attribute values, e.g., `attr.source = c('.numberLines', 'startFrom="11"')`.

- `render`: (`knitr::knit_print`; `function(x, options, ...)`) A function to print the visible values in a chunk. The value passed to the first argument of this function (i.e., `x`) is the value evaluated from each expression in the chunk. The list of current chunk options is passed to the argument `options`. This function is expected to return a character string. For more information, check out the package vignette about custom chunk rendering: `vignette('knit_print', package = 'knitr')`.

- `split`: (`FALSE`; logical) Whether to split the output into separate files and include them in LaTeX by `\input{}` or HTML by `<iframe></iframe>`. This option only works for `.Rnw`, `.Rtex`, and `.Rhtml` documents.

Code decoration

- `tidy`: (`FALSE`) Whether to reformat the R code. Other possible values are as follows:
  - `TRUE` (equivalent to `tidy = 'formatR'`): Call the function `formatR::tidy_source()` to reformat the code.
  - `'styler'`: Use `styler::style_text()` to reformat the code.
  - A custom function of the form `function(code, ...) {}` to return the reformatted code.
  - If reformatting fails, the original R code will not be changed (with a warning).

- `tidy.opts`: (`NULL`; list) A list of options to be passed to the function determined by the `tidy` option, e.g., `tidy.opts = list(blank = FALSE, width.cutoff = 60)` for `tidy = 'formatR'` to remove blank lines and try to cut the code lines at 60 characters.

- `prompt`: (`FALSE`; logical) Whether to add the prompt characters in the R code. See `prompt` and `continue` on the help page `?base::options`. Note that adding prompts can make it difficult for readers to copy R code from the output, so `prompt = FALSE` may be a better choice. This option may not work well when the chunk option `engine` is not `R` (#1274).

- `comment`: (`'##'`; character) The prefix to be added before each line of the text output. By default, the text output is commented out by `##`, so if readers want to copy and run the source code from the output document, they can select and copy everything from the chunk, since the text output is masked in comments (and will be ignored when running the copied text). Set `comment = ''` remove the default `##`.

- `highlight`: (`TRUE`; logical) Whether to syntax highlight the source code.

- `class.source`: (NULL; character) Class names for source code blocks in the output document. Similar to the `class.*` options for output such as `class.output`.

- `attr.source`: (`NULL`; character) Attributes for source code blocks. Similar to the `attr.*` options for output such as `attr.output`.

- `lang`: (`NULL`; character) The language name of a code chunk. By default, the language name is the engine name, e.g., `r`. This option is primarily for syntax highlighting of Markdown-based output documents.

- `size`: (`'normalsize'`; character) Font size of the chunk output from `.Rnw` documents. See this page for possible sizes.

- `background`: (`'#F7F7F7'`; character) Background color of the chunk output of `.Rnw` documents.

- `indent`: (character) A string to be added to each line of the chunk output. Typically it consists of white spaces. This option is assumed to be read-only, and **knitr** sets its value while parsing the document. For example, for the chunk below, `indent` is a character string of two spaces:

  ~~~
  ```{r}
  rnorm(10)
  ```
  ~~~

Cache

- `cache`: (`FALSE`; logical) Whether to cache a code chunk. When evaluating code chunks for the second time, the cached chunks are skipped (unless they have been modified), but the objects created in these chunks are loaded from previously saved databases (`.rdb` and `.rdx` files), and these files are saved when a chunk is evaluated for the first time, or when cached files are not found (e.g., you may have removed them by hand). Note that the filename consists of the chunk label with an MD5 digest of the R code and chunk options of the code chunk, which means any changes in the chunk will produce a different MD5 digest, and hence invalidate the cache. See more information on this page.

- `cache.path`: (`'cache/'`; character) A prefix to be used to generate the paths of cache files. For R Markdown, the default value is based on the input filename, e.g., the cache paths for the chunk with the label `FOO` in the file `INPUT.Rmd` will be of the form `INPUT_cache/FOO_*.*`.

- `cache.vars`: (`NULL`; character) A vector of variable names to be saved in the cache database. By default, all variables created in the current chunks are identified and saved, but you may want to manually specify the variables to be saved, because the automatic detection of variables may not be robust, or you may want to save only a subset of variables.

- `cache.globals`: (`NULL`; character) A vector of the names of variables that are not created from the current chunk. This option is mainly for `autodep = TRUE` to work more precisely—a chunk `B` depends on chunk `A` when any of `B`'s global variables are `A`'s local variables. In case the automatic detection of global variables (i.e., when `cache.globals = NULL` or `TRUE`) in a chunk fails, you may manually specify the names of global variables via this option (see #1403 for an example). In addition, `cache.globals = FALSE` means detecting all variables in a code chunk, no matter if they are global or local variables.

- `cache.lazy`: (`TRUE`; logical) Whether to `lazyLoad()` or directly `load()` objects. For very large objects, lazyloading may not work, so `cache.lazy = FALSE` may be desirable (see #572).

- `cache.comments`: (`NULL`; logical) If `FALSE`, changing comments in R code chunks will not invalidate the cache database.

- `cache.rebuild`: (`FALSE`; logical) If `TRUE`, reevaluate the chunk even if the cache does not need to be invalidated. This can be useful when you want to conditionally invalidate the cache, e.g., `cache.rebuild = !file.exists("some-file")` can rebuild the chunk when `some-file` does not exist (see #238).

- `dependson`: (`NULL`; character or numeric) A character vector of chunk labels to specify which other chunks this chunk depends on. This option applies to cached chunks only—sometimes the objects in a cached chunk may depend on other cached chunks, so when other chunks are changed, this chunk must be updated accordingly.
    - If `dependson` is a numeric vector, it means the indices of chunk labels, e.g., `dependson = 1` means this chunk depends on the first chunk in the document, and `dependson = c(-1, -2)` means it depends on the previous two chunks (negative indices stand for numbers of chunks before this chunk, and note they are always relative to the current chunk).
    - Please note this option does not work when set as a global chunk option via `opts_chunk$set()`; it must be set as a local chunk option.

- `autodep`: (`FALSE`; logical) Whether to analyze dependencies among chunks automatically by detecting global variables in the code (may not be reliable), so `dependson` does not need to be set explicitly.

Plots

- `fig.path`: (`'figure/'`; character) A prefix to be used to generate figure file paths. `fig.path` and chunk labels are concatenated to generate the full paths. It may contain a directory like `figure/prefix-`; the directory will be created if it does not exist.

- `fig.keep`: (`'high'`; character) How plots in chunks should be kept. Possible values are as follows:

    - `high`: Only keep high-level plots (merge low-level changes into high-level plots).
    - `none`: Discard all plots.
    - `all`: Keep all plots (low-level plot changes may produce new plots).
    - `first`: Only keep the first plot.
    - `last`: Only keep the last plot.
    - If set to a numeric vector, the values are indices of (low-level) plots to keep.

    Low-level plotting commands include `lines()` and `points()`, etc. To better understand `fig.keep`, consider the following chunk:

    ~~~
    ```{r, test-plot}
    plot(1)         # high-level plot
    abline(0, 1)    # low-level change
    plot(rnorm(10)) # high-level plot
    # many low-level changes in a loop (a single R expression)
    for(i in 1:10) {
        abline(v = i, lty = 2)
    }
    ```
    ~~~

    Normally this produces 2 plots in the output (because `fig.keep = 'high'`). For `fig.keep = 'none'`, no plots will be saved. For `fig.keep = 'all'`, 4 plots are saved. For `fig.keep = 'first'`, the plot produced by `plot(1)` is saved. For `fig.keep = 'last'`, the last plot with 10 vertical lines is saved.

- `fig.show`: (`'asis'`; character) How to show/arrange the plots. Possible values are as follows:

    - `asis`: Show plots exactly in places where they were generated (as if the code were run in an R terminal).
    - `hold`: Hold all plots and output them at the end of a code chunk.
    - `animate`: Concatenate all plots into an animation if there are multiple plots in a chunk.
    - `hide`: Generate plot files but hide them in the output document.

- `dev`: (`'pdf'` for LaTeX output and `'png'` for HTML/Markdown; character) The graphical device to generate plot files. All graphics devices in base R and those in **Cairo**, **svglite**, **ragg**, and **tikzDevice** are supported, e.g., `pdf`, `png`, `svg`, `jpeg`, `tiff`, `cairo_pdf`, `CairoJPEG`, `CairoPNG`, svglite, gridSVG, ragg_png, tikz, and so on. See `names(knitr::auto_exts)` for the full list. Besides these devices, you can also provide a character string that is the name of a function of the form `function(filename, width, height)`. The units for the image size are *always* inches (even for bitmap devices, in which DPI is used to convert between pixels and inches).

The chunk options `dev`, `fig.ext`, `fig.width`, `fig.height`, and `dpi` can be vectors (shorter ones will be recycled), and they are vectorized over every single plot in a code chunk to create multiple copies of the same plot. For example, `dev = c('pdf', 'png')` will create a PDF and a PNG file for the same plot. Note that when the plot files to be created have the same extension, you must use the `fig.ext` option to specify different filename suffixes, otherwise latter plot files will overwrite previous ones. For example, when `dev = 'png'` and `fig.width = c(10, 6)`, you can generate two PNG images with different widths with `fig.ext = c('1.png', '2.png')`.

- `dev.args`: (`NULL`; list) More arguments to be passed to the device, e.g., `dev.args = list(bg = 'yellow', pointsize = 10)` for `dev = 'png'`. This option depends on the specific device (see the device documentation). When `dev` contains multiple devices, `dev.args` can be a list of lists of arguments, and each list of arguments is passed to each individual device, e.g., `dev = c('pdf', 'tiff'), dev.args = list(pdf = list(colormodel = 'cmyk', useDingats = TRUE), tiff = list(compression = 'lzw'))`.
- `fig.ext`: (`NULL`; character) File extension of the figure output. If `NULL`, it will be derived from the graphical device; see `knitr:::auto_exts` for details.
- `dpi`: (`72`; numeric) The DPI (dots per inch) for bitmap devices (`dpi * inches = pixels`).
- `fig.width`, `fig.height`: (both are `7`; numeric) Width and height of the plot (in inches), to be used in the graphics device.
- `fig.asp`: (`NULL`; numeric) The aspect ratio of the plot, i.e., the ratio of height/width. When `fig.asp` is specified, the height of a plot (the chunk option `fig.height`) is calculated from `fig.width * fig.asp`.
- `fig.dim`: (`NULL`; numeric) A numeric vector of length 2 to provide `fig.width` and `fig.height`, e.g., `fig.dim = c(5, 7)` is a shorthand of `fig.width = 5, fig.height = 7`. If both `fig.asp` and `fig.dim` are provided, `fig.asp` will be ignored (with a warning).
- `out.width`, `out.height`: (`NULL`; character) Width and height of the plot in the output document, which can be different with its physical `fig.width` and `fig.height`, i.e., plots can be scaled in the output document. Depending on the output format, these two options can take

special values. For example, for LaTeX output, they can be `.8\\linewidth`, `3in`, or `8cm`; for HTML, they may be `300px`. For `.Rnw` documents, the default value for `out.width` will be changed to `\\maxwidth`, which is defined on this page. It can also be a percentage, e.g., `'40%'` will be translated to `0.4\linewidth` when the output format is LaTeX.

- `out.extra`: (`NULL`; character) Extra options for figures. It can be an arbitrary string, to be inserted in `\includegraphics[]` in LaTeX output (e.g., `out.extra = 'angle=90'` to rotate the figure by 90 degrees), or `<img />` in HTML output (e.g., `out.extra = 'style="border:5px solid orange;"'`).

- `fig.retina`: (`1`; numeric) This option only applies to HTML output. For Retina displays, setting this option to a ratio (usually 2) will change the chunk option `dpi` to `dpi * fig.retina`, and `out.width` to `fig.width * dpi / fig.retina` internally. For example, the physical size of an image is doubled, and its display size is halved when `fig.retina = 2`.

- `resize.width`, `resize.height`: (`NULL`; character) The width and height to be used in `\resizebox{}{}` in LaTeX output. These two options are not needed unless you want to resize TikZ graphics, because there is no natural way to do it. However, according to the **tikzDevice** authors, TikZ graphics are not meant to be resized, to maintain consistency in style with other text in LaTeX. If only one of them is `NULL`, `!` will be used (read the documentation of **graphicx** if you do not understand this).

- `fig.align`: (`'default'`; character) Alignment of figures in the output document. Possible values are `default`, `left`, `right`, and `center`. The default is not to make any alignment adjustments.

- `fig.link`: (`NULL`; character) A link to be added onto the figure.

- `fig.env`: (`'figure'`; character) The LaTeX environment for figures, e.g., you may set `fig.env = 'marginfigure'` to get `\begin{marginfigure}`. This option requires `fig.cap` be specified.

- `fig.cap`: (`NULL`; character) A figure caption.

- `fig.alt`: (`NULL`; character) The alternative text to be used in the `alt` attribute of the `<img>` tags of figures in HTML output. By default, the chunk option `fig.cap` will be used as the alternative text if provided.

- `fig.scap`: (`NULL`; character) A short caption. This option is only meaningful to LaTeX output. A short caption is inserted in `\caption[]`, and usually displayed in the "List of Figures" of a PDF document.

- `fig.lp`: (`'fig:'`; character) A label prefix for the figure label to be inserted in `\label{}`. The actual label is made by concatenating this prefix and the chunk label, e.g., the figure label for ` ```{r, foo-plot} ` will be `fig:foo-plot` by default. Note that the insertion of `\label{}` depends on the chunk being rendered as LaTeX (see this issue).

- `fig.id`: (`NULL`; logical) When `TRUE`, automatic IDs will be assigned to images generated from code chunks in R Markdown, i.e., images will be written in HTML code like `<img id="..." />`. The default ID consists of `fig.lp` (a prefix), `label` (the chunk label), and `fig.cur` (the current figure number in the chunk). Any non-alphanumeric characters in the ID will be substituted with dashes, e.g., `'fig:hello world 1'` will become `'fig-hello-world-1'`. Alternatively, users can provide a function to generate the ID. The function takes the list of the current chunk options as the input and should return a character string, e.g., `fig.id = function(options) { paste0('img-', options$label, options$fig.cur) }`.

- `fig.pos`: (`''`; character) A character string for the figure position arrangement to be used in `\begin{figure}[]`.

- `fig.subcap`: (`NULL`) Captions for subfigures. When there are multiple plots in a chunk, and neither `fig.subcap` nor `fig.cap` is `NULL`, `\subfloat{}` will be used for individual plots (you need to add `\usepackage{subfig}` in the preamble). See 067-graphics-options.Rnw for an example.

- `fig.ncol`: (`NULL`; integer) The number of columns of subfigures; see this issue for examples (note that `fig.ncol` and `fig.sep` only work for LaTeX output).

- `fig.sep`: (`NULL`; character) A character vector of separators to be inserted among subfigures. When `fig.ncol` is specified, `fig.sep` defaults to a character vector of which every N-th element is `\newline` (where `N` is the number of columns), e.g., `fig.ncol = 3` means `fig.sep = c('', '', '\\newline', '', '', '\\newline', '', ...)`. The *i*-th separator is added after the *i*-th subfigure, except when the length of `fig.sep` is greater than the number of subfigures, in which case the first `fig.sep` element is added before the first subfigure, and the (*i*+*1*)-th element is added after the *i*-th subfigure.

- `fig.process`: (`NULL`; function) A function to post-process figure files. It should take the path of a figure file, and return the (new) path of the figure to be inserted in the output. If the function contains the `options` argument, the list of chunk options will be passed to this argument.

- `fig.showtext`: (`NULL`; logical) If `TRUE`, call `showtext::showtext_begin()` before drawing plots. See the documentation of the **showtext** package for details.

- `external`: (`TRUE`; logical) Whether to externalize tikz graphics (pre-compile tikz graphics to PDF). It is only used for the `tikz()` device in the **tikzDevice** package (i.e., when `dev='tikz'`), and it can save time for LaTeX compilation.

- `sanitize`: (`FALSE`; character) Whether to sanitize tikz graphics (escape special LaTeX characters). See the documentation of the **tikzDevice** package.

There are two hidden options that are not designed to be set by users: `fig.cur` (the current figure number or index when there are multiple figures), and `fig.num` (the total number of figures in a chunk). The purpose of these two options is to help **knitr** deal with the filenames of multiple figures as well as animations. In some cases, we can make use of them to write animations into the output using plot files that are saved manually (see the graphics manual for examples).

Animation

- `interval`: (`1`; numeric) Time interval (number of seconds) between animation frames.

- `animation.hook`: (`knitr::hook_ffmpeg_html`; function or character) A hook function to create animations in HTML output; the default hook uses FFmpeg to convert images to a WebM video.

  - Another hook function is `knitr::hook_gifski` based on the **gifski** package to create GIF animations.

  - This option can also take a character string `'ffmpeg'` or `'gifski'` as a shorthand of the corresponding hook function, e.g., `animation.hook = 'gifski'` means `animation.hook = knitr::hook_gifski`.

  - `aniopts`: (`'controls,loop'`; character) Extra options for animations; see the documentation of the LaTeX **animate** package.

- `ffmpeg.bitrate` (`1M`; character) To be passed to the `-b:v` argument of FFmpeg to control the quality of WebM videos.

- `ffmpeg.format` (`webm`; character) The video format of FFmpeg, i.e., the filename extension of the video.

Code chunk

- `code`: (`NULL`; character) If provided, it will override the code in the current chunk. This allows us to programmatically insert code into the current chunk. For example, `code = readLines('test.R')` will use the content of the file `test.R` as the code for the current chunk.

- `file`: (`NULL`; character) If provided, it will override the `code` option by reading the chunk content from external files. A chunk option `file = "test.R"` is equivalent to `code = xfun::read_all("test.R")`.

- `ref.label`: (`NULL`; character) A character vector of labels of the chunks from which the code of the current chunk is inherited (see the demo for chunk references). If the vector is wrapped in `I()` and the chunk option `opts.label` is not set, it means that the current chunk will also inherit the chunk options (in addition to the code) of the referenced chunks. See the chunk option `opts.label` for more information on inheriting chunk options.

Child documents

- `child`: (`NULL`; character) A character vector of paths of child documents to be knitted and input into the main document.

Language engines

- `engine`: (`'R'`; character) The language name of the code chunk. Possible values can be found in `names(knitr::knit_engines$get())`, e.g., `python`, `sql`, `julia`, `bash`, and `c`, etc. The object `knitr::knit_engines` can be used to set up engines for other languages. The demo page contains examples of different engines.

- `engine.path`: (`NULL`; character) The path to the executable of the `engine`. This option makes it possible to use alternative executables in your system, e.g., the default `python` may be at `/usr/bin/python`, and you may set `engine.path = '~/anaconda/bin/python'` to use a different version of Python.

  `engine.path` can also be a list of paths, which makes it possible to set different engine paths for different engines, e.g.,

  ```r
  knitr::opts_chunk$set(engine.path = list(
    python = '~/anaconda/bin/python',
    ruby = '/usr/local/bin/ruby'
  ))
  ```

  The names of the list correspond to the names of the engines.

- `engine.opts`: (`NULL`; character) Additional arguments passed to the engines. At the chunk level, the option can be specified as a string or a list of options.

~~~
```{bash, engine.opts='-l'}
echo $PATH
```


```{cat, engine.opts = list(file = "my_custom.css")}
h2 {
  color: blue;
}
```
~~~

At the global level, it could be a list of strings named by engines. Like `engine.path`, this is useful to template arguments via `knitr::opts_chunk$set()`.

```
knitr::opts_chunk$set(engine.opts = list(
  perl = '-Mstrict -Mwarnings',
  bash = '-o errexit'
))
```

Each engine as its own use of `engine.opts` and defines specific options. One should consult the documentation of each engine. Some examples are in the R Markdown Cookbook for the `cat` engine and the `sass/scss` engine.

Option templates

- `opts.label`: (`NULL`; character) This option provides a mechanism to inherit chunk options from either the option template `knitr::opts_template` (see `?knitr::opts_template`) or other code chunks. It takes a character vector of labels. For each label in the vector, **knitr** will first try to find chunk options set in `knitr::opts_template` with this label, and if found, apply these chunk options to the current chunk. Then try to find another code chunk with this label (called the "referenced code chunk") in the document, and if found, also apply its chunk options to the current chunk.

  The precedence of chunk options is: local chunk options > referenced code chunk options > `knitr::opts_template` > `knitr::opts_chunk`.

  The special value `opts.label = TRUE` means `opts.label = ref.label`, i.e., to inherit chunk options from chunks referenced by the `ref.label` option. See the example #121 in the knitr-examples repository for demos of various usage of `ref.label` and `opts.label`.

Extracting source code

- `purl`: (`TRUE`; logical) When running `knitr::purl()` to extract source code from a source document, whether to include or exclude a certain code chunk.

Other chunk options

- `R.options`: (`NULL`; list) Local R options for a code chunk. These options are set temporarily via `options()` before the code chunk, and restored after the chunk.

Package Options

The package options can be changed using the object `knitr::opts_knit` (*not to be confused with `knitr::opts_chunk`*). For example:

```
knitr::opts_knit$set(progress = TRUE, verbose = TRUE)
```

See `?knitr::opts_knit` for the alternative approach to setting package options using the R base function `options()`.

Available package options are as follows:

- `aliases`: (`NULL`; character) A named character vector to specify the aliases of chunk options, e.g., `c(h = 'fig.height', w = 'fig.width')` tells **knitr** that the chunk option `h` really means `fig.height`, and `w` is an alias for `fig.width`. This option can be used to save some typing effort for long option names.
- `base.dir`: (`NULL`; character) An absolute directory under which the plots are generated.
- `base.url`: (`NULL`; character) The base URL of images on HTML pages.
- `concordance`: (`FALSE`; logical) Whether to write a concordance file to map the output line numbers to the input line numbers. This enables one to navigate from the output to the input, and can be helpful especially when a TeX error occurs. This feature is only for `.Rnw` documents, and implemented in RStudio.
- `eval.after`: (`c('fig.cap', 'fig.scap', 'fig.alt')`; character) A character vector of option names. These options will be evaluated *after* a chunk has been evaluated, and all other options will be evaluated before a chunk. For example, for `fig.cap = paste('p-value is',`

`t.test(x)$p.value`), it will be evaluated after the chunk according to the value of `x` if `eval.after = 'fig.cap'`.

- `global.par`: (`FALSE`; logical) If `TRUE`, the `par()` settings from the previous code chunk will be preserved and applied to the next code chunk (of course, this only applies to base R graphics). By default, **knitr** opens a new graphical device to record plots and close it after evaluating the code, so `par()` settings will be discarded.

- `header`: (`NULL`; character) The text to be inserted into the output document before the document begins (e.g., after `\documentclass{article}` in LaTeX, or after `<head>` in HTML). This is useful for defining commands and styles in the LaTeX preamble or HTML header. The beginning of document is found using the pattern defined in `knitr::knit_patterns$get('document.begin')`. This option is only for `.Rnw` and `.Rhtml` documents.

- `label.prefix`: (`c(table = 'tab:')`; character) The prefix for labels. Currently only the prefix for table labels generated by `knitr::kable()` is supported.

- `latex.options.color`, `latex.options.graphicx` (`NULL`): Options for the LaTeX packages **color** and **graphicx**, respectively. These options are only for `.Rnw` documents.

- `latex.tilde` (`NULL`; character): The LaTeX command string for the tilde character in the syntax highlighted chunk output from `.Rnw` documents (see the issue #1992 for examples).

- `out.format`: (`NULL`; character) Possible values are `latex`, `sweave`, `html`, `markdown`, and `jekyll`. It will be automatically determined based on the input file, and this option will affect the set of hooks to be set (see `?knitr::render_latex` for example). Note this option has to be set *before* `knitr::knit()` runs (it will not work if you set it inside the document).

- `progress`: (`TRUE`; logical) Whether to display a progress bar when running `knitr::knit()`.

- `root.dir`: (`NULL`; character) The root directory when evaluating code chunks. If `NULL`, the directory of the input document will be used.

- `self.contained`: (`TRUE`; logical) Whether the output document should be self-contained (TeX styles to be written in the `.tex` document, and CSS styles to be written in the `.html` document). This option only applies to `.Rnw` and `.Rhtml` documents.

- `unnamed.chunk.label`: (`unnamed-chunk`; character) The label prefix for unnamed chunks.

- `upload.fun`: (`identity`; function) A function that takes a file path, processes the file, and returns a character string when the output format is HTML or Markdown. Typically, it is a function to upload an image and return the link to the image, e.g., `knitr::opts_knit$set(upload.fun = knitr::imgur_upload)` can upload a file to imgur.com (see `?knitr::imgur_upload`).

- `verbose`: (`FALSE`; logical) Whether to show verbose information (e.g., R code in each chunk and message logs), or only show chunk labels and options.

Global R Options

Global R options are set with `options()` in base R. Below is a list of options that affect the behavior of **knitr**:

- `knitr.bib.prefix`: (`R-`; character) The prefix for keys of bibliography entries generated by `knitr::write_bib()`.

- `knitr.child.warning`: (`TRUE`; logical) When a code chunk uses the `child` option to include child documents, a warning will be issued if the code chunk is not empty (because the code will be ignored). This option can be set to `FALSE` to suppress the warning.

- `knitr.digits.signif`: (`FALSE`; logical) When formatting numeric values from inline R expressions, whether to use `format()` (`TRUE`) or `round()` (`FALSE`). The former means that the global option `digits` (set via `option(digits =)`) specifies the number of significant digits. The latter means the option `digits` specifies the number of decimal places.

- `knitr.duplicate.label`: (`NULL`) If set to `"allow"`, duplicate chunk labels will be allowed in the same document.

- `knitr.include.graphics.ext`: (`FALSE`; logical) Whether to keep the filename extension when including a plot file path in `\includegraphics{}` for LaTeX output.

- `knitr.progress.simple`: (`NULL`; logical) Whether to show the (text) bar in the progress output.

- `knitr.progress.fun`: (`knitr:::txt_pb`; function) A function of the form `function(total, labels) {}`. This function should take arguments `total` (the total number of chunks) and `labels` (the vector of chunk labels), create a progress bar, and return a list of two methods: `list(update = function(i) {}, done = function() {})`. The `update()` method takes `i` (the index of the current chunk) as the input and updates the progress bar. The `done()` method closes the progress bar.

Below is an example of using the **cli** progress bar:

```
function(total, labels) {
  id = cli::cli_progress_bar(
    total = total, .auto_close = FALSE
  )
  list(
    update = function(i) {
      cli::cli_progress_update(id = id)
    },
    done = function() {
      cli::cli_process_done(id)
    }
  )
}
```

And below is an example of using a Windows progress bar (which works only on Windows, obviously):

```
function(total, labels) {
  pb = winProgressBar("Knitting...", max = total)
  list(
    update = function(i) {
      setWinProgressBar(pb, i, label = labels[i])
    },
    done = function() {
      close(pb)
    }
  )
}
```

- `knitr.progress.linenums`: (`FALSE`; logical) Whether to show line numbers in the progress bar. By default, only chunk labels are shown.
- `knitr.progress.output`: (`""`; character or connection) For the default text progress bar in **knitr**, this option can be used to specify the output target of the progress bar. By default, the

progress is written to `stdout()`. If you prefer using `stderr` instead, you can set this option to `stderr()`.

- `knitr.purl.inline`: (`FALSE`; logical) Whether to include inline R code in the `knitr::purl()` output.
- `knitr.svg.object`: (`FALSE`; logical) If `TRUE`, `svg` plots will be embedded as raw XML (`<svg>`) in self-contained HTML output, and they will be included in `<object>` tags if the HTML output is not self-contained. If `FALSE`, the `<img>` tag will be used.

← Frequently Asked Questions ∅ →