Package ‘devtools’

April 11, 2020

Title  Tools to Make Developing R Packages Easier

Version  2.3.0

Description  Collection of package development tools.

License  GPL (>= 2)


BugReports  https://github.com/r-lib/devtools/issues

Depends  R (>= 3.0.2), usethis (>= 1.6.0)

Imports  callr (>= 3.4.3), cli (>= 2.0.2), covr (>= 3.5.0), crayon (>= 1.3.4), desc (>= 1.2.0), digest (>= 0.6.25), DT (>= 0.13), ellipsis (>= 0.3.0), glue (>= 1.4.0), git2r (>= 0.26.1), htr (>= 1.4.1), jsonlite (>= 1.6.1), memoise (>= 1.1.0), pkgbuild (>= 1.0.6), pkgload (>= 1.0.2), rcmdcheck (>= 1.3.3), remotes (>= 2.1.1), rlang (>= 0.4.5), roxygen2 (>= 7.1.0), rstudioapi (>= 0.11), rversions (>= 2.0.1), sessioninfo (>= 1.1.1), stats, testthat (>= 2.3.2), tools, utils, withr (>= 2.1.2)

Suggests  BiocManager (>= 1.30.10), curl (>= 4.3), foghorn (>= 1.1.5), gmailer (>= 1.0.0), knitr (>= 1.28), lintr (>= 2.0.1), mockery (>= 0.4.2), pingr (>= 2.0.0), MASS, pkgdown (>= 1.5.1), rhub (>= 1.1.1), rmarkdown (>= 2.1), spelling (>= 2.1)

VignetteBuilder  knitr

Encoding  UTF-8

RoxygenNote  7.1.0

Language  en-US

NeedsCompilation  no

Author  Hadley Wickham [aut],
        Jim Hester [aut, cre],
        Winston Chang [aut],
        RStudio [cph],
        R Core team [ctb] (Some namespace and vignette code extracted from base R)

Maintainer  Jim Hester <jim. hyster@rstudio.com>
Repository CRAN
Date/Publication 2020-04-10 23:20:11 UTC

R topics documented:

bash ................................................................. 3
build ............................................................... 3
build_manual ..................................................... 4
build_rmd .......................................................... 5
build_site .......................................................... 5
build_vignettes ................................................... 6
check ............................................................... 7
check_failures .................................................... 9
check_man .......................................................... 10
check_rhub .......................................................... 11
check_win ........................................................... 12
clean_vignettes ................................................... 13
create .............................................................. 14
devtools ............................................................. 14
dev_mode ............................................................ 15
dev_sitrep ........................................................... 16
document ............................................................ 17
install ............................................................... 17
install_deps .......................................................... 19
lint ................................................................. 20
load_all ............................................................. 21
missing_s3 .......................................................... 23
package_file ........................................................ 23
release ............................................................. 24
reload ............................................................... 25
revdep .............................................................. 25
run_examples ....................................................... 27
save_all ............................................................ 28
show_news ........................................................... 28
source_gist .......................................................... 29
source_url .......................................................... 30
spell_check .......................................................... 31
test ................................................................. 31
uninstall ............................................................ 32
wd ................................................................. 33

Index 34
bash

Open bash shell in package directory.

Usage

bash(pkg = ".")

Arguments

pkg The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

build

Build package

Description

Building converts a package source directory into a single bundled file. If `binary = FALSE` this creates a `tar.gz` package that can be installed on any platform, provided they have a full development environment (although packages without source code can typically be installed out of the box). If `binary = TRUE`, the package will have a platform specific extension (e.g. `.zip` for windows), and will only be installable on the current platform, but no development environment is needed.

Usage

build(
    pkg = ".",
    path = NULL,
    binary = FALSE,
    vignettes = TRUE,
    manual = FALSE,
    args = NULL,
    quiet = FALSE,
    ...
)
Arguments

pkg The package to use, can be a file path to the package or a package object. See as.package() for more information.

path Path in which to produce package. If NULL, defaults to the parent directory of the package.

binary Produce a binary (--binary) or source ( --no-manual --no-resave-data) version of the package.

vignettes For source packages: if FALSE, don’t build PDF vignettes (--no-build-vignettes) or manual (--no-manual).

manual For source packages: if FALSE, don’t build PDF vignettes (--no-build-vignettes) or manual (--no-manual).

args An optional character vector of additional command line arguments to be passed to R CMD build if binary = FALSE, or R CMD install if binary = TRUE.

quiet if TRUE suppresses output from this function.

... Additional arguments passed to pkgbuild::build.

Value

a string giving the location (including file name) of the built package

Description

Create package pdf manual

Usage

build_manual(pkg = ".", path = NULL)

Arguments

pkg The package to use, can be a file path to the package or a package object. See as.package() for more information.

path path in which to produce package manual. If NULL, defaults to the parent directory of the package.

See Also

Rd2pdf()
build_rmd  

Build a Rmarkdown files package

Description

build_rmd() is a wrapper around rmarkdown::render() that first installs a temporary copy of the package, and then renders each .Rmd in a clean R session. build_readme() locates your README.Rmd and builds it into a README.md

Usage

build_rmd(files, path = ".", output_options = list(), ..., quiet = TRUE)

build_readme(path = ".", quiet = TRUE, ...)

Arguments

files The Rmarkdown files to be rendered.
path path to the package to build the readme.
output_options List of output options that can override the options specified in metadata (e.g. could be used to force self_contained or mathjax = "local"). Note that this is only valid when the output format is read from metadata (i.e. not a custom format object passed to output_format).
... additional arguments passed to rmarkdown::render()
quiet If TRUE, suppress output.

build_site

Execute pkgdown build_site in a package

Description

build_site() is a shortcut for pkgdown::build_site(), it generates the static HTML documentation.

Usage

build_site(path = ".", quiet = TRUE, ...)

Arguments

path path to the package to build the static HTML.
quiet If TRUE, suppress output.
... additional arguments passed to pkgdown::build_site()
build_vignettes  

Build package vignettes.

**Description**

Builds package vignettes using the same algorithm that R CMD build does. This means including non-Sweave vignettes, using makefiles (if present), and copying over extra files. The files are copied in the 'doc' directory and an vignette index is created in 'Meta/vignette.rds', as they would be in a built package. 'doc' and 'Meta' are added to .Rbuildignore, so will not be included in the built package. These files can be checked into version control, so they can be viewed with browseVignettes() and vignette() if the package has been loaded with load_all() without needing to re-build them locally.

**Usage**

```r
build_vignettes(
  pkg = ".", dependencies = "VignetteBuilder",
  clean = TRUE,
  upgrade = "never",
  quiet = TRUE,
  install = TRUE,
  keep_md = TRUE
)
```

**Arguments**

- **pkg** The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
- **dependencies** Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector. TRUE is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". NA is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. FALSE is shorthand for no dependencies (i.e. just check this package, not its dependencies).
- **clean** Remove all files generated by the build, even if there were copies there before.
- **upgrade** One of "default", "ask", "always", or "never". "default" respects the value of the R_REMOTES_UPGRADE environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". TRUE and FALSE are also accepted and correspond to "always" and "never" respectively.
- **quiet** If TRUE, suppresses most output. Set to FALSE if you need to debug.
- **install** If TRUE, install the package before building vignettes.
If TRUE, move md intermediates as well as rendered outputs. Most useful when using the keep_md YAML option for Rmarkdown outputs. See https://bookdown.org/yihui/rmarkdown/html-document.html#keeping-markdown.

See Also

clean_vignettes() to remove the pdfs in ‘doc’ created from vignettes

clean_vignettes() to remove build tex/pdf files.

---

**check**

*Build and check a package, cleaning up automatically on success.*

---

**Description**

check automatically builds and checks a source package, using all known best practices. check_built checks an already built package.

**Usage**

```r
check(
  pkg = ".",
  document = NA,
  build_args = NULL,
  ...,
  manual = FALSE,
  cran = TRUE,
  remote = FALSE,
  incoming = remote,
  force_suggests = FALSE,
  run_dont_test = FALSE,
  args = "--timings",
  env_vars = c(NOT_CRAN = "true"),
  quiet = FALSE,
  check_dir = tempdir(),
  cleanup = TRUE,
  vignettes = TRUE,
  error_on = c("never", "error", "warning", "note")
)
```

```r
check_built(
  path = NULL,
  cran = TRUE,
  remote = FALSE,
  incoming = remote,
  force_suggests = FALSE,
  run_dont_test = FALSE,
  manual = FALSE,
  ...,
)```
check

```r
args = "--timings",
env_vars = NULL,
check_dir = tempdir(),
quiet = FALSE,
error_on = c("never", "error", "warning", "note")
```

**Arguments**

- **pkg**
  - The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

- **document**
  - If `NA` and the package uses roxygen2, will rerun `document()` prior to checking. Use `TRUE` and `FALSE` to override this default.

- **build_args**
  - Additional arguments passed to `R CMD build` ...

- **manual**
  - If `FALSE`, don’t build and check manual (`--no-manual`).

- **cran**
  - If `TRUE` (the default), check using the same settings as CRAN uses.

- **remote**
  - Sets `_R_CHECK_CRAN_INCOMING_REMOTE_` env var. If `TRUE`, performs a number of CRAN incoming checks that require remote access.

- **incoming**
  - Sets `_R_CHECK_CRAN_INCOMING_` env var. If `TRUE`, performs a number of CRAN incoming checks.

- **force_suggests**
  - Sets `_R_CHECK_FORCE_SUGGESTS_`. If `FALSE` (the default), check will proceed even if all suggested packages aren’t found.

- **run_dont_test**
  - Sets `--run-donttest` so that tests surrounded in `\dontest{}` are also tested. This is important for CRAN submission.

- **args**
  - Additional arguments passed to `R CMD check`

- **env_vars**
  - Environment variables set during `R CMD check`

- **quiet**
  - If `TRUE` suppresses output from this function.

- **check_dir**
  - The directory in which the package is checked compatibility. `args = "--output=/foo/bar"` can be used to change the check directory.

- **cleanup**
  - Deprecated.

- **vignettes**
  - If `FALSE`, do not build or check vignettes, equivalent to using `args = '--ignore-vignettes'` and `build_args = '--no-build-vignettes'`.

- **error_on**
  - Whether to throw an error on `R CMD check` failures. Note that the check is always completed (unless a timeout happens), and the error is only thrown after completion. If "never", then no errors are thrown. If "error", then only ERROR failures generate errors. If "warning", then WARNING failures generate errors as well. If "note", then any check failure generated an error.

- **path**
  - Path to built package.
Details

Passing R CMD check is essential if you want to submit your package to CRAN: you must not have any ERRORs or WARNINGs, and you want to ensure that there are as few NOTEs as possible. If you are not submitting to CRAN, at least ensure that there are no ERRORs or WARNINGs: these typically represent serious problems.

check automatically builds a package before calling check_built as this is the recommended way to check packages. Note that this process runs in an independent realisation of R, so nothing in your current workspace will affect the process.

Value

An object containing errors, warnings, and notes.

Environment variables

Devtools does its best to set up an environment that combines best practices with how check works on CRAN. This includes:

- The standard environment variables set by devtools: r_env_vars(). Of particular note for package tests is the NOT_CRAN env var which lets you know that your tests are not running on CRAN, and hence can take a reasonable amount of time.
- Debugging flags for the compiler, set by compiler_flags(FALSE).
- If aspell is found _R_CHECK_CRAN_INCOMING_USE_ASPELL_ is set to TRUE. If no spell checker is installed, a warning is issued.)
- env vars set by arguments incoming, remote and force_suggests

See Also

release() if you want to send the checked package to CRAN.

check_failures Parses R CMD check log file for ERRORs, WARNINGs and NOTEs

Description

Extracts check messages from the 00check.log file generated by R CMD check.

Usage

check_failures(path, error = TRUE, warning = TRUE, note = TRUE)

Arguments

path check path, e.g., value of the check_dir argument in a call to check()
error, warning, note logical, indicates if errors, warnings and/or notes should be returned
check_man

Value

a character vector with the relevant messages, can have length zero if no messages are found

See Also

check()

check_man

Check documentation, as R CMD check does.

Description

This function attempts to run the documentation related checks in the same way that R CMD check does. Unfortunately it can’t run them all because some tests require the package to be loaded, and the way they attempt to load the code conflicts with how devtools does it.

Usage

check_man(pkg = ".")

Arguments

pkg

The package to use, can be a file path to the package or a package object. See as.package() for more information.

Value

Nothing. This function is called purely for it’s side effects: if

Examples

## Not run:
check_man("mypkg")

## End(Not run)
check_rhub  

Run CRAN checks for package on R-hub

Description

It runs build() on the package, with the arguments specified in args, and then submits it to the R-hub builder at https://builder.r-hub.io. The interactive option controls whether the function waits for the check output. Regardless, after the check is complete, R-hub sends an email with the results to the package maintainer.

Usage

```r
check_rhub(
  pkg = ".", 
  platforms = NULL,
  email = NULL, 
  interactive = TRUE, 
  build_args = NULL, 
  ... 
)
```

Arguments

- **pkg**  
The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

- **platforms**  
R-hub platforms to run the check on. If NULL uses default list of CRAN checkers (one for each major platform, and one with extra checks if you have compiled code). You can also specify your own, see `rhub::platforms()` for a complete list.

- **email**  
email address to notify, defaults to the maintainer address in the package.

- **interactive**  
whether to show the status of the build interactively. R-hub will send an email to the package maintainer's email address, regardless of whether the check is interactive or not.

- **build_args**  
Arguments passed to R CMD build

- **...**  
extra arguments, passed to `rhub::check_for_cran()`.

Value

a rhub_check object.

About email validation on r-hub

To build and check R packages on R-hub, you need to validate your email address. This is because R-hub sends out emails about build results. See more at `rhub::validate_email()`. 
See Also

Other build functions: check_win()

---

check_win  
*Build windows binary package.*

Description

This function works by bundling source package, and then uploading to https://win-builder.r-project.org/. Once building is complete you’ll receive a link to the built package in the email address listed in the maintainer field. It usually takes around 30 minutes. As a side effect, win-build also runs R CMD check on the package, so check_win is also useful to check that your package is ok on windows.

Usage

```r
check_win_devel(
  pkg = ".",
  args = NULL,
  manual = TRUE,
  email = NULL,
  quiet = FALSE,
  ...;
)
```

```r
check_win_release(
  pkg = ".",
  args = NULL,
  manual = TRUE,
  email = NULL,
  quiet = FALSE,
  ...;
)
```

```r
check_win_oldrelease(
  pkg = ".",
  args = NULL,
  manual = TRUE,
  email = NULL,
  quiet = FALSE,
  ...;
)
```

Arguments

pkg  
The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
**clean_vignettes**

An optional character vector of additional command line arguments to be passed to `R CMD build` if `binary = FALSE`, or `R CMD install` if `binary = TRUE`.

For source packages: if `FALSE`, don't build PDF vignettes (``--no-build-vignettes``) or manual (``--no-manual``).

An alternative email to use, default `NULL` uses the package Maintainer's email.

If `TRUE`, suppresses output.

Additional arguments passed to `pkgbuild::build()`.

**Functions**

- `check_win_devel`: Check package on the development version of R.
- `check_win_release`: Check package on the release version of R.
- `check_win_oldrelease`: Check package on the previous major release version of R.

**See Also**

Other build functions: `check_rhub()`

---

**clean_vignettes**

Clean built vignettes.

**Description**

This uses a fairly rudimentary algorithm where any files in 'doc' with a name that exists in 'vignettes' are removed.

**Usage**

```r
clean_vignettes(pkg = ".")
```

**Arguments**

- `pkg`
  The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
create  

Create a package

Description
Create a package

Usage
create(path, ...)

Arguments
path  
A path. If it exists, it is used. If it does not exist, it is created, provided that the
parent path exists.

...  
Additional arguments passed to `usethis::create_package()`

Value
The path to the created package, invisibly.

devtools  

Package development tools for R.

Description
Collection of package development tools.

Package options
Devtools uses the following `options()` to configure behaviour:

- `devtools.path`: path to use for `dev_mode()`
- `devtools.name`: your name, used when signing draft emails.
- `devtools.install.args`: a string giving extra arguments passed to R CMD install by `install()`.
- `devtools.desc.author`: a string providing a default `Authors@R` string to be used in new
  `DESCRIPTION`'s. Should be a R code, and look like "Hadley Wickham <h.wickham@gmail.com>
  [aut,cre]". See `utils::as.person()` for more details.
- `devtools.desc.license`: a default license string to use for new packages.
- `devtools.desc.suggests`: a character vector listing packages to to add to suggests by de-
  faults for new packages.
- `devtools.desc`: a named list listing any other extra options to add to `DESCRIPTION`
dev_mode

Author(s)

Maintainer: Jim Hester <jim.hester@rstudio.com>

Authors:

• Hadley Wickham
• Winston Chang

Other contributors:

• RStudio [copyright holder]
• R Core team (Some namespace and vignette code extracted from base R) [contributor]

See Also

Useful links:

• https://devtools.r-lib.org/
• https://github.com/r-lib/devtools
• Report bugs at https://github.com/r-lib/devtools/issues

---

dev_mode  Activate and deactivate development mode.

Description

When activated, dev_mode creates a new library for storing installed packages. This new library
is automatically created when dev_mode is activated if it does not already exist. This allows you
to test development packages in a sandbox, without interfering with the other packages you have
installed.

Usage

dev_mode(on = NULL, path = getOption("devtools.path"))

Arguments

on           turn dev mode on (TRUE) or off (FALSE). If omitted will guess based on whether
or not path is in .libPaths()
path         directory to library.

Examples

## Not run:
dev_mode()
dev_mode()

dev_mode()

## End(Not run)
dev_sitrep  

Report package development situation

Description

dev_sitrep() reports

- If R is up to date
- If RStudio is up to date
- If compiler build tools are installed and available for use
- If devtools and its dependencies are up to date
- If the package’s dependencies are up to date

Call this function if things seem weird and you’re not sure what’s wrong or how to fix it. If this function returns no output everything should be ready for package development.

Usage

dev_sitrep(pkg = ".", debug = FALSE)

Arguments

pkg  The package to use, can be a file path to the package or a package object. See as.package() for more information.

debug  If TRUE, will print out extra information useful for debugging. If FALSE, it will use result cached from a previous run.

Value

A named list, with S3 class dev_sitrep (for printing purposes).

Examples

```r
## Not run:
dev_sitrep()

## End(Not run)
```
Use roxygen to document a package.

Description

This function is a wrapper for the roxygen2::roxygenize() function from the roxygen2 package. See the documentation and vignettes of that package to learn how to use roxygen.

Usage

document(pkg = ".", roclets = NULL, quiet = FALSE)

Arguments

pkg The package to use, can be a file path to the package or a package object. See as.package() for more information.

roclets Character vector of roclet names to use with package. The default, NULL, uses the roxygen roclets option, which defaults to c("collate","namespace","rd").

quiet if TRUE suppresses output from this function.

See Also

roxygen2::roxygenize(), browseVignettes("roxygen2")

Install a local development package.

Description

Uses R CMD INSTALL to install the package. Will also try to install dependencies of the package from CRAN, if they’re not already installed.

Usage

install(
  pkg = ".",
  reload = TRUE,
  quick = FALSE,
  build = !quick,
  args =getOption("devtools.install.args"),
  quiet = FALSE,
  dependencies = NA,
  upgrade = "ask",
  build_vignettes = FALSE,
  keep_source = getOption("keep.source.pkgs"),
install

force = FALSE,
...
)

Arguments

pkg       The package to use, can be a file path to the package or a package object. See as.package() for more information.
reload    if TRUE (the default), will automatically reload the package after installing.
quick     if TRUE skips docs, multiple-architectures, demos, and vignettes, to make installation as fast as possible.
build     if TRUE pkgbuild::build()s the package first: this ensures that the installation is completely clean, and prevents any binary artefacts (like '.o', '.so') from appearing in your local package directory, but is considerably slower, because every compile has to start from scratch.
args      An optional character vector of additional command line arguments to be passed to R CMD INSTALL. This defaults to the value of the option "devtools.install.args".
quiet     If TRUE, suppress output.
dependencies Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector.
            TRUE is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". NA is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. FALSE is shorthand for no dependencies (i.e. just check this package, not its dependencies).
upgrade   One of "default", "ask", "always", or "never". "default" respects the value of the R_REMOTES_UPGRADE environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". TRUE and FALSE are also accepted and correspond to "always" and "never" respectively.
build_vignettes if TRUE, will build vignettes. Normally it is build that’s responsible for creating vignettes; this argument makes sure vignettes are built even if a build never happens (i.e. because build = FALSE).
keep_source If TRUE will keep the srcrefs from an installed package. This is useful for debugging (especially inside of RStudio). It defaults to the option "keep.source.pkgs".
force     Force installation, even if the remote state has not changed since the previous install.
...
additional arguments passed to remotes::install_deps() when installing dependencies.

Details

If quick = TRUE, installation takes place using the current package directory. If you have compiled code, this means that artefacts of compilation will be created in the src/ directory. If you want
to avoid this, you can use `build = TRUE` to first build a package bundle and then install it from a temporary directory. This is slower, but keeps the source directory pristine.

If the package is loaded, it will be reloaded after installation. This is not always completely possible, see `reload()` for caveats.

To install a package in a non-default library, use `withr::with_libpaths()`.

See Also

`update_packages()` to update installed packages from the source location and `with_debug()` to install packages with debugging flags set.

Other package installation: `uninstall()`

---

### install_deps

**Install package dependencies if needed.**

Description

`install_deps()` will install the user dependencies needed to run the package. `install_dev_deps()` will also install the development dependencies needed to test and build the package.

Usage

```r
install_deps(
  pkg = ".",
  dependencies = NA,
  repos =getOption("repos"),
  type =getOption("pkgType"),
  upgrade = c("default", "ask", "always", "never"),
  quiet = FALSE,
  build = TRUE,
  build_opts = c("--no-resave-data", "--no-manual", " --no-build-vignettes"),
  ...
)
```

```r
install_dev_deps(
  pkg = ".",
  dependencies = TRUE,
  repos =getOption("repos"),
  type =getOption("pkgType"),
  upgrade = c("default", "ask", "always", "never"),
  quiet = FALSE,
  build = TRUE,
  build_opts = c("--no-resave-data", "--no-manual", " --no-build-vignettes"),
  ...
)
```
Arguments

pkg The package to use, can be a file path to the package or a package object. See \texttt{as.package()} for more information.

dependencies Which dependencies do you want to check? Can be a character vector (selecting from "Depends", "Imports", "LinkingTo", "Suggests", or "Enhances"), or a logical vector.

\texttt{TRUE} is shorthand for "Depends", "Imports", "LinkingTo" and "Suggests". \texttt{NA} is shorthand for "Depends", "Imports" and "LinkingTo" and is the default. \texttt{FALSE} is shorthand for no dependencies (i.e. just check this package, not its dependencies).

repos A character vector giving repositories to use.

type Type of package to update.

upgrade One of "default", "ask", "always", or "never". "default" respects the value of the \texttt{R_REMOTES_UPGRADE} environment variable if set, and falls back to "ask" if unset. "ask" prompts the user for which out of date packages to upgrade. For non-interactive sessions "ask" is equivalent to "always". \texttt{TRUE} and \texttt{FALSE} are also accepted and correspond to "always" and "never" respectively.

quiet If \texttt{TRUE}, suppress output.

build if \texttt{TRUE} \texttt{pkgbuild::build()}s the package first: this ensures that the installation is completely clean, and prevents any binary artefacts (like ".o", ".so") from appearing in your local package directory, but is considerably slower, because every compile has to start from scratch.

build_opts Options to pass to \texttt{R CMD build}, only used when \texttt{build}

\ldots additional arguments passed to \texttt{remotes::install_deps()} when installing dependencies.

Examples

\begin{verbatim}
## Not run: install_deps(".")
\end{verbatim}

\section*{Description}

The default linters correspond to the style guide at \url{http://r-pkgs.had.co.nz/r.html#style}, however it is possible to override any or all of them using the \texttt{linters} parameter.

\section*{Usage}

\begin{verbatim}
lint(pkg = ".", cache = TRUE, ...)\end{verbatim}
Arguments

pkg

The package to use, can be a file path to the package or a package object. See as.package() for more information.

cache

store the lint results so repeated lints of the same content use the previous results.

... addition arguments passed to lintr::lint_package()

Details

The lintr cache is by default stored in ~/.R/lintr_cache/ (this can be configured by setting options(lintr.cache_directory)). It can be cleared by calling lintr::clear_cache().

See Also

lintr::lint_package(), lintr::lint()
helpers if TRUE loads testthat test helpers.
quiet if TRUE suppresses output from this function.
... Additional arguments passed to pkgload::load_all().

Details
Currently load_all:

- Loads all data files in data/. See load_data() for more details.
- Sources all R files in the R directory, storing results in environment that behaves like a regular package namespace. See below and load_code() for more details.
- Compiles any C, C++, or Fortran code in the src/ directory and connects the generated DLL into R. See compile_dll() for more details.
- Runs .onAttach(), .onLoad() and .onUnload() functions at the correct times.
- If you use testthat, will load all test helpers so you can access them interactively. devtools sets the DEVTOOLS_LOAD environment variable to "true" to let you check whether the helpers are run during package loading.

Namespaces
The namespace environment <namespace:pkgname>, is a child of the imports environment, which has the name attribute imports:pkgname. It is in turn a child of <namespace:base>, which is a child of the global environment. (There is also a copy of the base namespace that is a child of the empty environment.)

The package environment <package:pkgname> is an ancestor of the global environment. Normally when loading a package, the objects listed as exports in the NAMESPACE file are copied from the namespace to the package environment. However, load_all by default will copy all objects (not just the ones listed as exports) to the package environment. This is useful during development because it makes all objects easy to access.

To export only the objects listed as exports, use export_all=FALSE. This more closely simulates behavior when loading an installed package with library(), and can be useful for checking for missing exports.

Shim files
load_all also inserts shim functions into the imports environment of the loaded package. It presently adds a replacement version of system.file which returns different paths from base::system.file.
This is needed because installed and uninstalled package sources have different directory structures. Note that this is not a perfect replacement for base::system.file.

Examples
## Not run:
# Load the package in the current directory
load_all("./")

# Running again loads changed files
load_all("./")
With reset=TRUE, unload and reload the package for a clean start
load_all("./", TRUE)

With export_all=FALSE, only objects listed as exports in NAMESPACE
# are exported
load_all("./", export_all = FALSE)

## End(Not run)

---

**missing_s3**

*Find missing s3 exports.*

**Description**

The method is heuristic - looking for objs with a period in their name.

**Usage**

```r
missing_s3(pkg = ".")
```

**Arguments**

- `pkg` The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

---

**package_file**

*Find file in a package.*

**Description**

It always starts by walking up the path until it finds the root directory, i.e. a directory containing DESCRIPTION. If it cannot find the root directory, or it can’t find the specified path, it will throw an error.

**Usage**

```r
package_file(..., path = ".")
```

**Arguments**

- `...` Components of the path.
- `path` Place to start search for package directory.
release

Examples

```r
## Not run:
package_file("figures", "figure_1")
## End(Not run)
```

release

Release package to CRAN.

Description

Run automated and manual tests, then post package to CRAN.

Usage

```r
release(pkg = ".", check = FALSE, args = NULL)
```

Arguments

- `pkg` The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
- `check` if `TRUE`, run checking, otherwise omit it. This is useful if you've just checked your package and you're ready to release it.
- `args` An optional character vector of additional command line arguments to be passed to R CMD build.

Details

The package release process will:

- Confirm that the package passes R CMD check on relevant platforms
- Confirm that important files are up-to-date
- Build the package
- Submit the package to CRAN, using comments in "cran-comments.md"

You can add arbitrary extra questions by defining an (un-exported) function called `release_questions()` that returns a character vector of additional questions to ask.

You also need to read the CRAN repository policy at [https://cran.r-project.org/web/packages/policies.html](https://cran.r-project.org/web/packages/policies.html) and make sure you're in line with the policies. `release` tries to automate as many of polices as possible, but it's impossible to be completely comprehensive, and they do change in between releases of devtools.

See Also

`usethis::use_release_issue()` to create a checklist of release tasks that you can use in addition to or in place of `release`. 
Description

This attempts to unload and reload an *installed* package. If the package is not loaded already, it does nothing. It’s not always possible to cleanly unload a package: see the caveats in `unload()` for some of the potential failure points. If in doubt, restart R and reload the package with `library()`.

Usage

```r
reload(pkg = ".", quiet = FALSE)
```

Arguments

- `pkg` The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
- `quiet` if TRUE suppresses output from this function.

See Also

- `load_all()` to load a package for interactive development.

Examples

```r
## Not run:
# Reload package that is in current directory
reload("."

# Reload package that is in ./ggplot2/
reload("ggplot2/

# Can use inst() to find the package path
# This will reload the installed ggplot2 package
reload(pkgload::inst("ggplot2"))

## End(Not run)
```

Description

Tools to check and notify maintainers of all CRAN and Bioconductor packages that depend on the specified package.
Usage

revdep(
  pkg,
  dependencies = c("Depends", "Imports", "Suggests", "LinkingTo"),
  recursive = FALSE,
  ignore = NULL,
  bioconductor = FALSE
)

revdep_maintainers(pkg = ".")

Arguments

pkg  Package name. This is unlike most devtools packages which take a path because
you might want to determine dependencies for a package that you don’t have
installed. If omitted, defaults to the name of the current package.

dependencies  A character vector listing the types of dependencies to follow.
recursive  If TRUE look for full set of recursive dependencies.
ignore  A character vector of package names to ignore. These packages will not appear
in returned vector.
bioconductor  If TRUE also look for dependencies amongst Bioconductor packages.

Details

The first run in a session will be time-consuming because it must download all package metadata
from CRAN and Bioconductor. Subsequent runs will be faster.

See Also

The revdepcheck package can be used to run R CMD check on all reverse dependencies.

Examples

## Not run:
revdep("ggplot2")

revdep("ggplot2", ignore = c("xkcd", "zoo"))

## End(Not run)
**run_examples**

Run all examples in a package.

### Description

One of the most frustrating parts of R CMD check is getting all of your examples to pass - whenever one fails you need to fix the problem and then restart the whole process. This function makes it a little easier by making it possible to run all examples from an R function.

### Usage

```r
run_examples(
  pkg = ".",
  start = NULL,
  show = TRUE,
  test = FALSE,
  run = FALSE,
  fresh = FALSE,
  document = TRUE
)
```

### Arguments

- **pkg**: The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
- **start**: Where to start running the examples: this can either be the name of Rd file to start with (with or without extensions), or a topic name. If omitted, will start with the (lexicographically) first file. This is useful if you have a lot of examples and don’t want to rerun them every time you fix a problem.
- **show**: DEPRECATED.
- **test**: if TRUE, code in `\donttest{}` will be commented out. If FALSE, code in `\testonly{}` will be commented out.
- **run**: if FALSE, code in `\dontrun{}` will be commented out.
- **fresh**: if TRUE, will be run in a fresh R session. This has the advantage that there’s no way the examples can depend on anything in the current session, but interactive code (like `browser()`) won’t work.
- **document**: if TRUE, `document()` will be run to ensure examples are updated before running them.
save_all

Save all documents in an active IDE session.

Description

Helper function wrapping IDE-specific calls to save all documents in the active session. In this form, callers of `save_all()` don’t need to execute any IDE-specific code. This function can be extended to include other IDE implementations of their equivalent `rstdioapi::documentSaveAll()` methods.

Usage

```
save_all()
```

show_news

Show package news

Description

Show package news

Usage

```
show_news(pkg = ".", latest = TRUE, ...)
```

Arguments

- **pkg**: The package to use, can be a file path to the package or a package object. See `as.package()` for more information.
- **latest**: if TRUE, only show the news for the most recent version.
- **...**: other arguments passed on to `news`
source_gist

Run a script on gist

Description

“Gist is a simple way to share snippets and pastes with others. All gists are git repositories, so they are automatically versioned, forkable and usable as a git repository.” https://gist.github.com/

Usage

source_gist(id, ..., filename = NULL, sha1 = NULL, quiet = FALSE)

Arguments

id

either full url (character), gist ID (numeric or character of numeric).

...

other options passed to source()

filename

if there is more than one R file in the gist, which one to source (filename ending
in `.R`)? Default NULL will source the first file.

sha1

The SHA-1 hash of the file at the remote URL. This is highly recommend as it
prevents you from accidentally running code that’s not what you expect. See
source_url() for more information on using a SHA-1 hash.

quiet

if FALSE, the default, prints informative messages.

See Also

source_url()

Examples

## Not run:
# You can run gists given their id
source_gist(6872663)
source_gist("6872663")

# Or their html url
source_gist("https://gist.github.com/hadley/6872663")
source_gist("gist.github.com/hadley/6872663")

# It's highly recommend that you run source_gist with the optional
# sha1 argument - this will throw an error if the file has changed since
# you first ran it
source_gist(6872663, sha1 = "54f1db27e60")
# Wrong hash will result in error
source_gist(6872663, sha1 = "54f1db27e61")

# You can specify a particular R file in the gist
source_gist(6872663, filename = "hi.r")
source_gist(6872663, filename = "hi.r", sha1 = "54f1db27e60")
### End(Not run)

**source_url**

*Run a script through some protocols such as http, https, ftp, etc.*

**Description**

If a SHA-1 hash is specified with the `sha1` argument, then this function will check the SHA-1 hash of the downloaded file to make sure it matches the expected value, and throw an error if it does not match. If the SHA-1 hash is not specified, it will print a message displaying the hash of the downloaded file. The purpose of this is to improve security when running remotely-hosted code; if you have a hash of the file, you can be sure that it has not changed. For convenience, it is possible to use a truncated SHA1 hash, down to 6 characters, but keep in mind that a truncated hash won’t be as secure as the full hash.

**Usage**

```r
source_url(url, ..., sha1 = NULL)
```

**Arguments**

- `url` *url*
- `...` other options passed to `source()`
- `sha1` The (prefix of the) SHA-1 hash of the file at the remote URL.

**See Also**

- `source_gist()`

**Examples**

```r
## Not run:

source_url("https://gist.github.com/hadley/6872663/raw/hi.r")

# With a hash, to make sure the remote file hasn't changed
source_url("https://gist.github.com/hadley/6872663/raw/hi.r",
  sha1 = "54f1db27e60bb7e0486d785604909b49e8feef9f9")

# With a truncated hash
source_url("https://gist.github.com/hadley/6872663/raw/hi.r",
  sha1 = "54f1db27e60")

## End(Not run)
```
spell_check

**spell_check** *Spelling checking*

**Description**

Runs a spell check on text fields in the package description file, manual pages, and optionally vignettes. Wraps the `spelling` package.

**Usage**

```r
spell_check(pkg = ".", vignettes = TRUE, use_wordlist = TRUE)
```

**Arguments**

- **pkg**
  - The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

- **vignettes**
  - Also check all `rmd` and `rnw` files in the pkg `vignettes` folder

- **use_wordlist**
  - Ignore words in the package `WORDLIST` file

---

**test**

*Execute test_that tests in a package.*

**Description**

`test()` is a shortcut for `testthat::test_dir()`, it runs all of a package’s tests. `test_file` runs `test()` on the active file. `test_coverage()` computes test coverage for your package. It is a shortcut for `covr::package_coverage()` and `covr::report()`. `test_coverage_file()` computes test coverage for the active file. Is a shortcut for `covr::file_coverage()` and `covr::report()`.

**Usage**

```r
test(pkg = ".", filter = NULL, stop_on_failure = FALSE, export_all = TRUE, ...)
```

```r
test_coverage(pkg = ".", show_report = interactive(), ...)
```

```r
uses_testthat(pkg = ".")
```

```r
test_file(file = find_active_file(), ...)
```

```r
test_coverage_file(
  file = find_active_file(),
  filter = TRUE,
  show_report = interactive(),
  export_all = TRUE,
  ...
)
```

---
uninstall

Arguments

pkg  The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

filter  If not NULL, only tests with file names matching this regular expression will be executed. Matching be performed on the file name after it has been stripped of "test-" and ".R".

stop_on_failure  If TRUE, throw an error if any tests fail.

For historical reasons, the default value of stop_on_failure is TRUE for test_package() and test_check() but FALSE for test_dir(), so if you're calling test_dir() you may want to consider explicitly setting stop_on_failure = TRUE.

export_all  If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file.

...  additional arguments passed to testthat::test_dir() and covr::package_coverage()

show_report  Show the test coverage report.

file  One or more source or test files. If a source file the corresponding test file will be run. The default is to use the active file in RStudio (if available).

Description

Uninstall a local development package.

Usage

uninstall(pkg = ".", unload = TRUE, quiet = FALSE, lib = .libPaths()[[1]])

Arguments

pkg  The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

unload  if TRUE (the default), will automatically unload the package prior to uninstalling.

quiet  If TRUE, suppress output.

lib  a character vector giving the library directories to remove the packages from. If missing, defaults to the first element in `.libPaths()`.

See Also

`with_debug()` to install packages with debugging flags set.

Other package installation: `install()`
wd | Set working directory.

**Description**

Set working directory.

**Usage**

wd(pkg = ".", path = "")

**Arguments**

- **pkg**
  The package to use, can be a file path to the package or a package object. See `as.package()` for more information.

- **path**
  path within package. Leave empty to change working directory to package directory.
Index

*Topic programming
  build_vignettes, 6
run_examples, 27
.libPaths, 32
.libPaths(), 15

as.package(), 3, 4, 6, 8, 10–13, 16–18, 20, 21, 23–25, 27, 28, 31–33

bash, 3
browser(), 27
build, 3
build(), 11
build_manual, 4
build_readme (build_rmd), 5
build_rmd, 5
build_site, 5
build_vignettes, 6

cHECK, 7
cHECK(), 9, 10
cHECK_built (CHECK), 7
cHECK_failures, 9
cHECK_man, 10
cHECK_rhub, 11, 13
cHECK_win, 12, 12
cHECK_win_devel (CHECK_win), 12
cHECK_win_oldrelease (CHECK_win), 12
cHECK_win_release (CHECK_win), 12
clean_vignettes, 13
clean_vignettes(), 7
compile_dll(), 22
compiler_flags, 9
covr::file_coverage(), 31
covr::package_coverage(), 31, 32
covr::report(), 31
create, 14

dev_mode, 15
dev_mode(), 14
dev_sitrep, 16
devtools, 14
devtools-package (devtools), 14
document, 17
document(), 8, 27

install, 17, 32
install(), 14
install_deps, 19
install_dev_deps (install_deps), 19

library(), 21, 22, 25
lint, 20
lintr::clear_cache(), 21
lintr::lint(), 21
lintr::lint_package(), 21
load_all, 21
load_all(), 25
load_code(), 22
load_data(), 22

missing_s3, 23

options(), 14

package_file, 23
pkgbuild::build, 4
pkgbuild::build(), 8, 13, 18, 20
pkgbuild::clean_dll(), 21
pkgdown::build_site(), 5
pkgload::load_all(), 22

r_env_vars(), 9
Rd2pdf(), 4
release, 24
release(), 9
reload, 25
reload(), 19
remotes::install_deps(), 18, 20
revdep, 25
revdep_maintainers (revdep), 25
INDEX

rhub::check_for_cran(), 11
rhub::platforms(), 11
rhub::validate_email(), 11
rmarkdown::render(), 5
roxygen2::roxygenize(), 17
run_examples, 27
save_all, 28
show_news, 28
source(), 29, 30
source_gist, 29
source_gist(), 30
source_url, 30
source_url(), 29
spell_check, 31
spelling, 31
test, 31
test_coverage(test), 31
test_coverage_file(test), 31
test_file(test), 31
testthat::test_dir(), 31, 32
uninstall, 19, 32
unload(), 21, 25
update_packages(), 19
usethis::create_package(), 14
usethis::use_release_issue(), 24
utils::as.person(), 14
wd, 33
with_debug(), 19, 32
withr::with_libpaths(), 19, 32
WORDLIST, 31