Package 'devtools'

October 13, 2022

Title Tools to Make Developing R Packages Easier

Version 2.4.5

Description Collection of package development tools.

License MIT + file LICENSE

URL https://devtools.r-lib.org/, https://github.com/r-lib/devtools

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

Depends R (>= 3.0.2), usethis (>= 2.1.6)

Imports cli (>= 3.3.0), desc (>= 1.4.1), ellipsis (>= 0.3.2), fs (>= 1.5.2), lifecycle (>= 1.0.1), memoise (>= 2.0.1), miniUI (>= 0.1.1.1), pkgbuild (>= 1.3.1), pkgdown (>= 2.0.6), pkgload (>= 1.3.0), profvis (>= 0.3.7), rcmdcheck (>= 1.4.0), remotes (>= 2.4.2), rlang (>= 1.0.4), roxygen2 (>= 7.2.1), rversions (>= 2.1.1), sessioninfo (>= 1.2.2), stats, testthat (>= 3.1.5), tools, urlchecker (>= 1.0.1), utils, withr (>= 2.5.0)

Suggests BiocManager (>= 1.30.18), callr (>= 3.7.1), covr (>= 3.5.1), curl (>= 4.3.2), digest (>= 0.6.29), DT (>= 0.23), foghorn (>= 1.4.2), gh (>= 1.3.0), gmailr (>= 1.0.1), httr (>= 1.4.3), knitr (>= 1.39), lintr (>= 3.0.0), MASS, mockery (>= 0.4.3), pingr (>= 2.0.1), rhub (>= 1.1.1), rmarkdown (>= 2.14), rstudioapi (>= 0.13), spelling (>= 2.2)

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Encoding UTF-8

Language en-US

RoxygenNote 7.2.1

Config/testthat/edition 3

NeedsCompilation no

Author Hadley Wickham [aut],
Jim Hester [aut],
Winston Chang [aut],
Jennifer Bryan [aut, cre] (<https://orcid.org/0000-0002-6983-2759>),
RStudio [cph, fnd]

Maintainer Jennifer Bryan <jenny@rstudio.com>

Repository CRAN

Date/Publication 2022-10-11 17:12:36 UTC

R topics documented:

bash *Open bash shell in package directory.*

Description

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, 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

Note

The default manual = FALSE is not suitable for a CRAN submission, which may require manual = TRUE. Even better, use submit_cran() or release().

---

build_manual *Create package pdf manual*

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

```
build_vignettes(
  pkg = ".",
  dependencies = "VignetteBuilder",
  clean = TRUE,
  upgrade = "never",
  quiet = FALSE,
  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). The value "soft" means the same as `TRUE`, "hard" means the same as `NA`. You can also specify dependencies from one or more additional fields, common ones include: • Config/Needs/website - for dependencies used in building the pkgdown site. • Config/Needs/coverage for dependencies used in calculating test coverage.
`clean`	Remove all files generated by the build, even if there were copies there before.
`upgrade`	Should package dependencies be upgraded? 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.
keep_md	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*

Description

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

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 R session, so nothing in your current workspace will affect the process. Under-the-hood, check() and check_built() rely on pkgbuild::build() and rcmdcheck::rcmdcheck().

Usage

```
check(
  pkg = ".",
  document = NULL,
  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 = NULL,
  cleanup = deprecated(),
  vignettes = TRUE,
  error_on = c("never", "error", "warning", "note")
)

check_built(
  path = NULL,
  cran = TRUE,
  remote = FALSE,
  incoming = remote,
  force_suggests = FALSE,
  run_dont_test = FALSE,
  manual = FALSE,
  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	By default (NULL) will document if your installed roxygen2 version matches the version declared in the DESCRIPTION file. Use TRUE or FALSE to override the default.
build_args	Additional arguments passed to R CMD build
...	Additional arguments passed on to pkgbuild::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. Because this is a moving target and is not uniform across all of CRAN's machine, this is on a "best effort" basis. It is more complicated than simply setting --as-cran.
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 examples surrounded in \donttest{} are also run. When cran = TRUE, this only affects R 3.6 and earlier; in R 4.0, code in \donttest{} is always run as part of CRAN submission.

args	Character vector of arguments to pass to R CMD check. Pass each argument as a single element of this character vector (do not use spaces to delimit arguments like you would in the shell). For example, to skip running of examples and tests, use args = c("--no-examples", "--no-tests") and not args = "--no-examples --no-tests". (Note that instead of the --output option you should use the check_dir argument, because --output cannot deal with spaces and other special characters on Windows.)
env_vars	Environment variables set during R CMD check
quiet	if TRUE suppresses output from this function.
check_dir	Path to a directory where the check is performed. If this is not NULL, then the a temporary directory is used, that is cleaned up when the returned object is garbage collected.
cleanup	**[Deprecated]** See check_dir for details.
vignettes	If FALSE, do not build or check vignettes, equivalent to using args = '--ignore-vignettes' and build_ = '–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. Its default can be modified with the RCMDCHECK_ERROR_ON environment variable. If that is not set, then "never" is used.
path	Path to built package.

Value

An object containing errors, warnings, notes, and more.

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_mac_release` *Check macOS package*

Description

This function works by bundling source package, and then uploading to https://mac.r-project.org/macbuilder/submit.html. This function returns a link to the page with the check results.

Usage

```
check_mac_release(
  pkg = ".",
  dep_pkgs = character(),
  args = NULL,
  manual = TRUE,
  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.
dep_pkgs	Additional custom dependencies to install prior to checking the package.
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`.
manual	Should the manual be built?
quiet	If `TRUE`, suppresses output.
...	Additional arguments passed to `pkgbuild::build()`.

Value

The url with the check results (invisibly)

See Also

Other build functions: `check_rhub()`, `check_win()`

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 no errors there will be no output.

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

```
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_mac_release(), 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

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

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

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.
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.
manual	Should the manual be built?
email	An alternative email to use, default NULL uses the package Maintainer's email.
quiet	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_mac_release()`, `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

```
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, ..., open = FALSE)
```

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()`
`open`	If `TRUE`, activates the new project: • If RStudio desktop, the package is opened in a new session. • If on RStudio server, the current RStudio project is activated. • Otherwise, the working directory and active project is changed.

Value

The path to the created package, invisibly.

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()

## 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

```
## Not run:
dev_sitrep()

## End(Not run)
```

---

document *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 *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 = "default",
  build_vignettes = FALSE,
  keep_source = getOption("keep.source.pkgs"),
  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).

The value "soft" means the same as TRUE, "hard" means the same as NA.

You can also specify dependencies from one or more additional fields, common ones include:

- Config/Needs/website - for dependencies used in building the pkgdown site.
- Config/Needs/coverage for dependencies used in calculating test coverage.

upgrade — Should package dependencies be upgraded? 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

```
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"),
  ...
)

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 `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). The value "soft" means the same as `TRUE`, "hard" means the same as `NA`.

You can also specify dependencies from one or more additional fields, common ones include:

- Config/Needs/website - for dependencies used in building the pkgdown site.
- Config/Needs/coverage for dependencies used in calculating test coverage.

`repos` A character vector giving repositories to use.

`type` Type of package to `update`.

`upgrade` Should package dependencies be upgraded? 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`, suppress output.

`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.

`build_opts` Options to pass to `R CMD build`, only used when `build` is `TRUE`.

`...` additional arguments passed to `remotes::install_deps()` when installing dependencies.

Examples

```
## Not run: install_deps(".")
```

---

lint *Lint all source files in a package*

Description

The default linters correspond to the style guide at https://style.tidyverse.org/, however it is possible to override any or all of them using the `linters` parameter.

Usage

```
lint(pkg = ".", cache = 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.

`cache` Store the lint results so repeated lints of the same content use the previous results. Consult the lintr package to learn more about its caching behaviour.

`...` Additional arguments passed to `lintr::lint_package()`.

See Also

lintr::lint_package(), lintr::lint()

---

load_all *Load complete package*

Description

load_all loads a package. It roughly simulates what happens when a package is installed and loaded with library().

Usage

```
load_all(
  path = ".",
  reset = TRUE,
  recompile = FALSE,
  export_all = TRUE,
  helpers = TRUE,
  quiet = FALSE,
  ...
)
```

Arguments

path	Path to a package, or within a package.
reset	clear package environment and reset file cache before loading any pieces of the package. This largely equivalent to running unload(), however the old namespaces are not completely removed and no .onUnload() hooks are called. Use reset = FALSE may be faster for large code bases, but is a significantly less accurate approximation.
recompile	DEPRECATED. force a recompile of DLL from source code, if present. This is equivalent to running pkgbuild::clean_dll() before load_all
export_all	If TRUE (the default), export all objects. If FALSE, export only the objects that are listed as exports in the NAMESPACE file.
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 pkgbuild::compile_dll() for more details.
- Loads any compiled translations in inst/po.
- 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.

is_loading() returns TRUE when it is called while load_all() is running. This may be useful e.g. in onLoad hooks.

Differences with loadNamespace() and library()

load_all() tries its best to reproduce the behaviour of loadNamespace() and library(). However it deviates from normal package loading in several ways.

- It doesn't install the package on disk, so system.file() has no way of determining the location of the development files. To work around this, pkgload installs its own version of system.file() on the search path to make it easier to use interactively while developing. However this definition is only visible to the global environment, not to the namespaces of third party packages.
  One workaround for other packages to see the development files of your package while you're developing with devtools is for them to use fs::path_package() instead of system.file().
- Whereas loadNamespace() and library() only load package dependencies when they are needed, load_all() loads all packages referenced in Imports at load time.

Namespaces

The namespace environment <namespace:pkgname>, is a child of the imports environment, which has the name attribute imports:pkgname. It is in turn is 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

```
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.

`release` *Release package to CRAN.*

Description

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

Usage

```
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' 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.

---

reload *Unload and reload package.*

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

```
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

```
## 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)
```

---

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

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

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.
run_donttest	if TRUE, do run \donttest sections in the Rd files.
run_dontrun	if TRUE, do run \dontrun sections in the Rd files.
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.
run, test	Deprecated, see run_dontrun and run_donttest above.

---

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 rstudioapi::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 speficy 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

```
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

```
## 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 = "54f1db27e60bb7e0486d785604909b49e8fef9f9")

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

## End(Not run)
```

spell_check *Spell checking*

Description

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

Usage

```
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 testthat tests in a package*

Description

- test() runs all tests in a package. It's a shortcut for testthat::test_dir()
- test_active_file() runs test() on the active file.
- test_coverage() computes test coverage for your package. It's a shortcut for covr::package_coverage() plus covr::report().
- test_coverage_active_file() computes test coverage for the active file. It's a shortcut for covr::file_coverage() plus covr::report().

Usage

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

test_active_file(file = find_active_file(), ...)

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

test_coverage_active_file(
  file = find_active_file(),
  filter = TRUE,
  show_report = interactive(),
  export_all = 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.
filter	If not NULL, only tests with file names matching this regular expression will be executed. Matching is performed on the file name after it's stripped of "test-" and ".R".
stop_on_failure	If TRUE, throw an error if any tests fail.
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 wrapped functions.
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).
show_report	Show the test coverage report.

---

uninstall *Uninstall a local development package*

Description

Uses remove.packages() to uninstall the package. To uninstall a package from a non-default library, use in combination with withr::with_libpaths().

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), ensures the package is unloaded, 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