Package ‘devtools’

October 26, 2018

Title Tools to Make Developing R Packages Easier

Version 2.0.1

Description Collection of package development tools.

License GPL (>= 2)

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

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

Depends R (>= 3.0.2)

Imports callr, cli, digest, git2r (>= 0.23.0), httr (>= 0.4),
        jsonlite, memoise (>= 1.0.0), pkgbuild (>= 1.0.2), pkgload (>= 1.0.1), rcmdcheck (>= 1.3.0), remotes (>= 2.0.0), rstudioapi
             (>= 0.6.0.9000), sessioninfo, stats, tools, usethis (>= 1.4.0),
        utils, withr

Suggests BiocInstaller, BiocManager, bitops, covr (>= 3.2.0), crayon,
        curl (>= 0.9), evaluate, foghorn (>= 1.0.1), gmailer (>= 0.7.0),
        knitr, lintr (>= 0.2.1), mockery, pingr, MASS, pkgdown, Rcpp
             (>= 0.10.0), rhub, rmarkdown, roxygen2 (>= 6.1.0), rversions,
        spelling (>= 1.1), testthat (>= 1.0.2), whisker

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.1.0

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 <james.hesters@rstudio.com>

Repository CRAN

Date/Publication 2018-10-26 21:10:41 UTC
R topics documented:

bash .................................................. 2
build .................................................. 3
build_manual ....................................... 4
build_readme ........................................ 4
build_site ........................................... 5
build_vignettes .................................... 5
check .................................................. 6
check_failures ...................................... 8
check_man .......................................... 9
check_rhub .......................................... 9
check_win .......................................... 10
clean_vignettes .................................... 11
devtools ............................................ 12
dev_mode ........................................... 13
document .......................................... 13
dr_devtools ........................................ 14
dr_github .......................................... 14
install ............................................. 15
install_deps ...................................... 16
lint .................................................. 17
load_all .......................................... 18
missing_s3 ......................................... 20
package_file ....................................... 20
release .......................................... 21
reload ............................................. 22
revdep .............................................. 22
run_examples ...................................... 23
save_all .......................................... 24
show_news ......................................... 25
source_gist ......................................... 25
source_url ......................................... 26
spell_check ......................................... 27
test .................................................. 27
uninstall .......................................... 28
wd .................................................... 29

Index 30

bash ..................................................
Open bash shell in package directory.

Description

Open bash shell in package directory.
Usage

bash(pkg = ".")

Arguments

pkg package description, can be path or package name. See as.package() for more information

Usage

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

Arguments

pkg Path to a package, or within a package.
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
build_manual

Create package pdf manual

Description

Create package pdf manual

Usage

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

Arguments

pkg  package description, can be path or package name. 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_readme

Build a Rmarkdown README for a package

Description

build_readme() is a wrapper around rmarkdown::render(), it generates the README.md from a README.Rmd file.

Usage

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

Arguments

path  path to the package to build the readme.
quiet  If TRUE, suppress output.
...  additional arguments passed to rmarkdown::render()
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 = TRUE, install = TRUE,
keep_md = TRUE)

Arguments

pkg package description, can be path or package name. 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 "ask", "always" or "never". "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, 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

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 = NULL, quiet = FALSE, check_dir = tempdir(), cleanup = 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  package description, can be path or package name. 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.
built_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.
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.
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 `check.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

Value

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

See Also

`check()`, `revdep_check()`
check_man

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`: package description, can be path or package name. See `as.package()` for more information.

Value
Nothing. This function is called purely for its 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](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, ...)
```
check_win

Arguments

pkg         package description, can be path or package name. 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.

e-mail      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.

...        extra arguments, passed to pkgbuild::build().

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().

check_win  Build windows binary package.

Description

This function works by bundling source package, and then uploading to http://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 build_win is also useful to check that your package is ok on windows.

Usage

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

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

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

10
Arguments

pkg  package description, can be path or package name. 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  For source packages: if FALSE, don’t build PDF vignettes (--no-build-vignettes) or manual (--no-manual).

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.

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  package description, can be path or package name. See as.package() for more information
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 defaults for new packages.
- `devtools.desc`: a named list listing any other extra options to add to ‘DESCRIPTION’

Author(s)

Maintainer: Jim Hester <james.hesters@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://github.com/r-lib/devtools](https://github.com/r-lib/devtools)
- Report bugs at [https://github.com/r-lib/devtools/issues](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**

```r
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**

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

## 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**

```r
document(pkg = ".", roclets = NULL)
```

**Arguments**

- `pkg` : package description, can be path or package name. See `as.package()` for more information.
- `roclets` : Character vector of roclet names to use with package. This defaults to NULL, which will use the roclets fields in the list provided in the Roxygen DESCRIPTION field. If none are specified, defaults to c("collate", "namespace", "rd").
See Also
  
  roxygen2::roxygenize(), browseVignettes("roxygen2")

---

### dr_devtools

**Diagnose potential devtools issues**

**Description**

This checks to make sure you’re using the latest release of R, the released version of RStudio (if you’re using it as your gui), and the latest version of devtools and its dependencies.

**Usage**

```r
dr_devtools()
```

**See Also**

Other doctors: `dr_github`

**Examples**

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

## End(Not run)
```

---

### dr_github

**Diagnose potential GitHub issues**

**Description**

Diagnose potential GitHub issues

**Usage**

```r
dr_github(path = ".")
```

**Arguments**

- `path` Path to repository to check. Defaults to current working directory

**See Also**

Other doctors: `dr_devtools`
install

Examples

```
  dr_github()
```

---

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"), force = FALSE, ...)
```

Arguments

- **pkg**
  - package description, can be path or package name. 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()` 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 "ask", "always" or "never". "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.
install_deps

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 `local` = TRUE).

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

By default, 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. `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.

Description

Install package dependencies if needed.

Usage

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

Arguments

pkg                  package description, can be path or package name. 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).
repos              A character vector giving repositories to use.

Arguments

upgrade     One of "ask", "always" or "never". "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 build the package before installing.

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

Examples

## Not run: install.deps(".")

lint (pkg = ".", cache = TRUE, ...)  

Description

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

Usage

    lint(pkg = ".", cache = TRUE, ...)
Arguments

pkg  package description, can be path or package name. See as.package() for more information

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

...  additional 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()

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 is equivalent to running unload() and is the default. Use reset = FALSE may be faster for large code bases, but is a significantly less accurate approximation.
recompile  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().
load_all

Details

Currently load_all:

- Loads all data files in data/. See \texttt{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 \texttt{load_code()} for more details.
- Compiles any C, C++, or Fortran code in the src/ directory and connects the generated DLL into R. See \texttt{compile_dll()} for more details.
- Runs \texttt{.onAttach()}, \texttt{.onLoad()} and \texttt{.onUnload()} functions at the correct times.
- If you use \texttt{testthat}, will load all test helpers so you can access them interactively. Devtools sets the \texttt{DEVTOOLS_LOAD} environment variable to “true” to let you check whether the helpers are run during package loading.

Namespaces

The namespace environment \texttt{<namespace:pkgname>}, is a child of the imports environment, which has the name attribute \texttt{imports:pkgname}. It is in turn a child of \texttt{<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 \texttt{<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 \texttt{export_all=FALSE}. This more closely simulates behavior when loading an installed package with \texttt{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 \texttt{system.file} which returns different paths from \texttt{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 \texttt{base::system.file}.

Examples

```r
## 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
```
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`  
  package description, can be path or package name. 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.

**Examples**

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

## End(Not run)
release

| 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  package description, can be path or package name. 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.

Guarantee

If a devtools bug causes one of the CRAN maintainers to treat you impolitely, I will personally send you a handwritten apology note. Please forward me the email and your address, and I’ll get a card in the mail.
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` package description, can be path or package name. 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.
run_examples

Usage

```r
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. This is used in `revdep_check()` to avoid packages with installation problems or extremely long check times.

- **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

- `revdep_check()` to run R CMD check on all reverse dependencies.

Examples

```r
## 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 = TRUE, fresh = FALSE, document = TRUE)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkg</td>
<td>package description, can be path or package name. See <code>as.package()</code> for more information</td>
</tr>
<tr>
<td>start</td>
<td>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.</td>
</tr>
<tr>
<td>show</td>
<td>DEPRECATED.</td>
</tr>
<tr>
<td>test</td>
<td>if TRUE, code in <code>\donttest{}</code> will be commented out. If FALSE, code in <code>\testonly{}</code> will be commented out.</td>
</tr>
<tr>
<td>run</td>
<td>if TRUE, code in <code>\dontrun{}</code> will be commented out.</td>
</tr>
<tr>
<td>fresh</td>
<td>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 <code>browser()</code>) won’t work.</td>
</tr>
<tr>
<td>document</td>
<td>if TRUE, <code>document()</code> will be run to ensure examples are updated before running them.</td>
</tr>
</tbody>
</table>

---

**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**

```r
to do
```

**Arguments**

- `pkg` package description, can be path or package name. 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, forking and usable as a git repository.”

https://gist.github.com/

**Usage**

```r
to do
```

**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

```r
source_gist(6872663)
sorce_gist("6872663")
```

Or their html url

```r
source_gist("https://gist.githubusercontent.com/hadley/6872663")
source_gist("gist.githubusercontent.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

```r
source_gist(6872663, sha1 = "54f1db27e60")
source_gist(6872663, sha1 = "54f1db27e61")
```

You can specify a particular R file in the gist

```r
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**

```r
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 = "54f1db27e60bb7e0486d78560490b49e8fe9f9")

# 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

<table>
<thead>
<tr>
<th>pkg</th>
<th>package description, can be path or package name. See as.package() for more information</th>
</tr>
</thead>
<tbody>
<tr>
<td>vignettes</td>
<td>also check all rmd and rnw files in the pkg vignettes folder</td>
</tr>
<tr>
<td>use_wordlist</td>
<td>ignore words in the package WORDLIST file</td>
</tr>
</tbody>
</table>

---

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, ...)
test_coverage(pkg = ".", show_report = interactive(), ...)
uses_testthat(pkg = ".")
test_file(file = find_active_file(), ...)
test_coverage_file(file = find_active_file(), filter = TRUE, show_report = interactive(), ...)
```

Arguments

- **pkg**: package description, can be path or package name. See `as.package()` for more information.
- **filter**: If not NULL, only tests with file names matching this regular expression will be executed. Matching will take on the file name after it has been stripped of “test-” and “.R”.
- ... 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).

---

**uninstall**

Uninstall a local development package.

Description

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

Usage

```r
uninstall(pkg = ".", unload = TRUE, quiet = FALSE, ...)
```

Arguments

- **pkg**: package description, can be path or package name. See `as.package()` for more information.
- **unload**: if TRUE (the default), will automatically unload the package prior to uninstalling.
- **quiet**: If TRUE, suppress output.
- ... additional arguments passed to `remove.packages()`.
Set working directory.

Usage

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

Arguments

pkg package description, can be path or package name. See as.package() for more information
path path within package. Leave empty to change working directory to package directory.

See Also

with_debug() to install packages with debugging flags set.
Other package installation: install
Index

*Topic programming
  build_vignettes, 5
  run_examples, 23
.libPaths(), 13
as.package(), 3–5, 7, 9–11, 13, 15, 17, 18, 20–22, 24, 25, 27–29
bash, 2
browser(), 24
build, 3
build(), 9
build_manual, 4
build_readme, 4
build_site, 5
build_vignettes, 5
check, 6
check(), 8
check_built (check), 6
check_failures, 8
check_man, 9
check_rhub, 9
check_win, 10
check_win_devel (check_win), 10
check_win_oldrelease (check_win), 10
check_win_release (check_win), 10
clean_vignettes, 11
clean_vignettes(), 6
compile_dll(), 19
compiler_flags, 8
covr::file_coverage(), 27
covr::package_coverage(), 27, 28
covr::report(), 27
dev_mode, 13
dev_mode(), 12
devtools, 12
devtools-package (devtools), 12
document, 13
document(), 7, 24
dr_devtools, 14, 14
dr_github, 14, 14
install, 15, 29
install(), 12
install_deps, 16
install_dev_deps (install_deps), 16
library(), 18, 19, 22
lint, 17
lintr::clear_cache(), 18
lintr::lint(), 18
lintr::lint_package(), 18
load_all, 18
load_all(), 22
load_code(), 19
load_data(), 19
missing_s3, 20
options(), 12
package_file, 20
pkgbuild::build, 3
pkgbuild::build(), 7, 10, 11, 15
pkgbuild::clean_dll(), 18
pkgdown::build_site(), 5
pkgload::load_all(), 18
r_env_vars(), 8
Rd2pdf(), 4
release, 21
release(), 8
reload, 22
reload(), 16
remotes::install_deps(), 16
remove.packages(), 28
revdep, 22
revdep_check(), 8, 23
revdep_maintainers (revdep), 22
INDEX

rhub::platforms(), 10
rhub::validate_email(), 10
rmarkdown::render(), 4
roxygen2::roxygenize(), 13, 14
run_examples, 23
save_all, 24
show_news, 25
source(), 25, 26
source_gist, 25
source_gist(), 26
source_url, 26
source_url(), 25
spell_check, 27
spelling, 27
test, 27
test_coverage(test), 27
test_coverage_file(test), 27
test_file(test), 27
testthat::test_dir(), 27, 28
uninstall, 16, 28
unload(), 18, 22
update_packages(), 16
uses_testthat(test), 27
utils::as.person(), 12
utils::install.packages(), 17
wd, 29
with_debug(), 16, 29
withr::with_libpaths(), 16, 28
WORDLIST, 27