Package ‘devtools’

June 27, 2018

Title Tools to Make Developing R Packages Easier
Version 1.13.6
Encoding UTF-8
Description Collection of package development tools.
URL https://github.com/r-lib/devtools
BugReports https://github.com/r-lib/devtools/issues
Depends R (>= 3.0.2)
Imports httr (>= 0.4), utils, tools, methods, memoise (>= 1.0.0),
whisker, digest, rstudioapi (>= 0.2.0), jsonlite, stats, git2r
(>= 0.11.0), withr
Suggests curl (>= 0.9), crayon, testthat (>= 1.0.2), BiocInstaller,
Rcpp (>= 0.10.0), MASS, rmarkdown, knitr, hunspell (>= 2.0),
lintr (>= 0.2.1), bitops, roxygen2 (>= 5.0.0), evaluate,
versions, covr, gmailr (>= 0.7.0)
License GPL (>= 2)
VignetteBuilder knitr
RoxygenNote 6.0.1
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. hyster@rstudio.com>
Repository CRAN
Date/Publication 2018-06-27 20:57:54 UTC
**R topics documented:**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>bash</td>
<td>3</td>
</tr>
<tr>
<td>build</td>
<td>4</td>
</tr>
<tr>
<td>build_github_devtools</td>
<td>5</td>
</tr>
<tr>
<td>build_vignettes</td>
<td>6</td>
</tr>
<tr>
<td>build_win</td>
<td>6</td>
</tr>
<tr>
<td>check</td>
<td>7</td>
</tr>
<tr>
<td>check_failures</td>
<td>9</td>
</tr>
<tr>
<td>check_man</td>
<td>9</td>
</tr>
<tr>
<td>clean_dll</td>
<td>10</td>
</tr>
<tr>
<td>clean_source</td>
<td>10</td>
</tr>
<tr>
<td>clean_vignettes</td>
<td>11</td>
</tr>
<tr>
<td>compiler_flags</td>
<td>11</td>
</tr>
<tr>
<td>compile_dll</td>
<td>12</td>
</tr>
<tr>
<td>create</td>
<td>12</td>
</tr>
<tr>
<td>create_description</td>
<td>13</td>
</tr>
<tr>
<td>devtools</td>
<td>14</td>
</tr>
<tr>
<td>dev_example</td>
<td>15</td>
</tr>
<tr>
<td>dev_help</td>
<td>15</td>
</tr>
<tr>
<td>dev_mode</td>
<td>16</td>
</tr>
<tr>
<td>document</td>
<td>17</td>
</tr>
<tr>
<td>dr_devtools</td>
<td>17</td>
</tr>
<tr>
<td>dr_github</td>
<td>18</td>
</tr>
<tr>
<td>eval_clean</td>
<td>18</td>
</tr>
<tr>
<td>github_pull</td>
<td>19</td>
</tr>
<tr>
<td>has_devel</td>
<td>20</td>
</tr>
<tr>
<td>help</td>
<td>20</td>
</tr>
<tr>
<td>infrastructure</td>
<td>21</td>
</tr>
<tr>
<td>inst</td>
<td>24</td>
</tr>
<tr>
<td>install</td>
<td>25</td>
</tr>
<tr>
<td>install_bio</td>
<td>26</td>
</tr>
<tr>
<td>install_bitbucket</td>
<td>27</td>
</tr>
<tr>
<td>install_cran</td>
<td>28</td>
</tr>
<tr>
<td>install_deps</td>
<td>29</td>
</tr>
<tr>
<td>install_git</td>
<td>30</td>
</tr>
<tr>
<td>install_github</td>
<td>31</td>
</tr>
<tr>
<td>install_local</td>
<td>32</td>
</tr>
<tr>
<td>install_svn</td>
<td>33</td>
</tr>
<tr>
<td>install_url</td>
<td>34</td>
</tr>
<tr>
<td>install_version</td>
<td>34</td>
</tr>
<tr>
<td>lint</td>
<td>35</td>
</tr>
<tr>
<td>load_all</td>
<td>36</td>
</tr>
<tr>
<td>load_code</td>
<td>38</td>
</tr>
<tr>
<td>load_data</td>
<td>38</td>
</tr>
<tr>
<td>load_dll</td>
<td>38</td>
</tr>
<tr>
<td>missing_s3</td>
<td>39</td>
</tr>
<tr>
<td>package_deps</td>
<td>39</td>
</tr>
</tbody>
</table>
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
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 install 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 package description, can be path or package name. 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.

Value

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

See Also

Other build functions: build_win
build_github_devtools  

Build the development version of devtools from GitHub.

Description

This function is especially useful for Windows users who want to upgrade their version of devtools to the development version hosted on GitHub. In Windows, it's not possible to upgrade devtools while the package is loaded because there is an open DLL, which in Windows can't be overwritten. This function allows you to build a binary package of the development version of devtools; then you can restart R (so that devtools isn’t loaded) and install the package.

Usage

build_github_devtools(outfile = NULL)

Arguments

outfile  
The name of the output file. If NULL (the default), it uses ./devtools.tgz (Mac and Linux), or ./devtools.zip (Windows).

Details

Mac and Linux users don’t need this function; they can use install_github to install devtools directly, without going through the separate build-restart-install steps.

This function requires a working development environment. On Windows, it needs https://cran.r-project.org/bin/windows/Rtools/.

Value

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

Examples

## Not run:
library(devtools)
build_github_devtools()

### Restart R before continuing ###
install.packages("./devtools.zip", repos = NULL)

# Remove the package after installation
unlink("./devtools.zip")

## End(Not run)
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. You need to ensure that these files are not included in the built package - ideally they should not be checked into source, or at least excluded with .Rbuildignore

Usage
build_vignettes(pkg = ".", dependencies = "VignetteBuilder")

Arguments
pkg    package description, can be path or package name. See as.package for more information
dependencies    logical indicating to also install uninstalled packages which this pkg depends on/links to/suggests. See argument dependencies of install.packages.

See Also
clean_vignettes to remove the pdfs in 'inst/doc' created from vignettes
clean_vignettes to remove build tex/pdf files.

build_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
build_win(pkg = ".", version = c("R-release", "R-devel"), args = NULL, quiet = FALSE)
check

Arguments
pkg package description, can be path or package name. See \as.package for more information.
version directory to upload to on the win-builder, controlling which version of R is used to build the package. Possible options are listed on \url{http://win-builder.r-project.org/}. Defaults to R-devel.
args An optional character vector of additional command line arguments to be passed to \texttt{R CMD build} if binary = FALSE, or \texttt{R CMD install} if binary = TRUE.
quiet if TRUE suppresses output from this function.

See Also
Other build functions: \texttt{build}

Description
check automatically builds and checks a source package, using all known best practices. \texttt{check\_built} checks an already built package.

Usage
\begin{verbatim}
check(pkg = ".", document = TRUE, build_args = NULL, ..., manual = FALSE, cran = TRUE, check_version = FALSE, force_suggests = FALSE, run_dont_test = FALSE, args = NULL, env_vars = NULL, quiet = FALSE, check_dir = tempdir(), cleanup = TRUE)
\end{verbatim}

\begin{verbatim}
check\_built(path = NULL, cran = TRUE, check_version = FALSE, force_suggests = FALSE, run_dont_test = FALSE, manual = FALSE, args = NULL, env_vars = NULL, check_dir = tempdir(), quiet = FALSE)
\end{verbatim}

Arguments
pkg package description, can be path or package name. See \texttt{as.package} for more information.
document if TRUE (the default), will update and check documentation before running formal check.
build_args Additional arguments passed to \texttt{R CMD build}...
manual If FALSE, don’t build and check manual (\texttt{--no-manual}).
cran if TRUE (the default), check using the same settings as CRAN uses.
check

check_version
Sets \_R\_CHECK\_CRAN\_INCOMING\_ env var. If TRUE, performs a number of checked related to version numbers of packages on CRAN.

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

cleanup
Deprecated.

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 check_version 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
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  package description, can be path or package name. See as.package for more information

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

```r
## Not run:
check_man("mypkg")

## End(Not run)
```

clean_dll

*Remove compiled objects from /src/ directory*

**Description**

Invisibly returns the names of the deleted files.

**Usage**

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

**Arguments**

- `pkg` package description, can be path or package name. See `as.package` for more information

**See Also**

- `compile_dll`

---

clean_source

*Sources an R file in a clean environment.*

**Description**

Opens up a fresh R environment and sources file, ensuring that it works independently of the current working environment.

**Usage**

```r
clean_source(path, quiet = FALSE)
```

**Arguments**

- `path` path to R script
- `quiet` If FALSE, the default, all input and output will be displayed, as if you’d copied and paste the code. If TRUE only the final result and the any explicitly printed output will be displayed.
clean_vignettes

Description
This uses a fairly rudimentary algorithm where any files in ‘inst/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

compiler_flags Default compiler flags used by devtools.

Description
These default flags enforce good coding practice by ensuring that CFLAGS and CXXFLAGS are set to -Wall -pedantic. These tests are run by cran and are generally considered to be good practice.

Usage
compiler_flags(debug = FALSE)

Arguments
debug If TRUE adds -g -O0 to all flags (Adding FFLAGS and FCFLAGS

Details
By default compile_dll is run with compiler_flags(TRUE), and check with compiler_flags(FALSE). If you want to avoid the possible performance penalty from the debug flags, install the package.

See Also
Other debugging flags: with_debug

Examples
compiler_flags()
compiler_flags(TRUE)
**compile_dll**

**Compile a .dll/.so from source.**

**Description**

`compile_dll` performs a fake `R CMD install so` code that works here should work with a regular install (and vice versa).

**Usage**

```r
compile_dll(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.

**Details**

During compilation, debug flags are set with `compiler_flags(TRUE)`. Invisibly returns the names of the DLL.

**Note**

If this is used to compile code that uses Rcpp, you will need to add the following line to your `Makevars` file so that it knows where to find the Rcpp headers:

```
/pkg_cppflags=＇$(R_HOME)/bin/Rscript -e 'Rcpp:::cxxflags'
```

**See Also**

- `clean_dll` to delete the compiled files.

---

**create**

**Creates a new package, following all devtools package conventions.**

**Description**

Similar to `package.skeleton`, except that it only creates the standard devtools directory structures; it doesn’t try and create source code and data files by inspecting the global environment.

**Usage**

```r
create(path = ".", description =getOption("devtools.desc"), check = FALSE,
       rstudio = TRUE, quiet = FALSE)

setup(path = ".", description =getOption("devtools.desc"), check = FALSE,
       rstudio = TRUE, quiet = FALSE)
```
create_description

Arguments

path location to create new package. The last component of the path will be used as the package name.
description list of description values to override default values or add additional values.
check if TRUE, will automatically run check
r studio Create an RStudio project file? (with use_rstudio)
quiet if FALSE, the default, prints informative messages.

Details

create requires that the directory doesn’t exist yet; it will be created but deleted upon failure. setup assumes an existing directory from which it will infer the package name.

See Also

Text with package.skeleton

Examples

## Not run:
# Create a package using all defaults:
path <- file.path(tempdir(), "myDefaultPackage")
create(path)

# Override a description attribute.
path <- file.path(tempdir(), "myCustomPackage")
my_description <- list("Maintainer" =
  
  
  "'Yoni Ben-Meshulam' <yoni@opower.com>"")
create(path, my_description)

## End(Not run)

create_description Create a default DESCRIPTION file for a package.

Description

Create a default DESCRIPTION file for a package.

Usage

create_description(path = ".", extra = getOption("devtools.desc"), quiet = FALSE)
Arguments

path  path to package root directory
extra  a named list of extra options to add to ‘DESCRIPTION’. Arguments that take a list
quiet  if TRUE, suppresses output from this function.

Details

To set the default author and licenses, set options `devtools.desc.author` and `devtools.desc.license`.
I use `options(devtools.desc.author = "Hadley Wickham <h.wickham@gmail.com> [aut, cre]", devtools.desc.license = "GPL-3.0")`.

---

**devtools**  Package development tools for R.

---

Description

Package development tools for R.

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 [as.person](https://github.com/hadley/devtools/wiki/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’
dev_example

Run a examples for an in-development function.

Description

Run a examples for an in-development function.

Usage

dev_example(topic)

Arguments

topic Name or topic (or name of Rd) file to run examples for

See Also

Other example functions: run_examples

Examples

## Not run:
# Runs installed example:
library("ggplot2")
example("ggplot")

# Runs development example:
load_all("ggplot2")
dev_example("ggplot")

## End(Not run)

dev_help

Read the in-development help for a package loaded with devtools.

Description

Note that this only renders a single documentation file, so that links to other files within the package won't work.

Usage

dev_help(topic, stage = "render", type = getOption("help_type"))
Arguments

- **topic**: name of help to search for.
- **stage**: at which stage ("build", "install", or "render") should \Sexpr macros be executed? This is only important if you're using \Sexpr macro's in your Rd files.
- **type**: of html to produce: "html" or "text". Defaults to your default documentation type.

Examples

```r
## Not run:
library("ggplot2")
help("ggplot") # loads installed documentation for ggplot

load_all("ggplot2")
development("ggplot") # loads development documentation for ggplot

## End(Not run)
```

---

`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 `roxygenize()` function from the roxygen2 package. See the documentation and vignettes of that package to learn how to use roxygen.

Usage

document(pkg = ".", clean = NULL, roclets = NULL, reload = TRUE)

Arguments

pkg
textpackage description, can be path or package name. See `as.package` for more information

clean, reload
Deprecated.

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

`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

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

dr_github(path = ".")

Arguments

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

See Also

Other doctors: dr_devtools

Examples


dr_github()

eval_clean

Evaluate code in a clean R session.

Description

Evaluate code in a clean R session.

Usage

eval_clean(expr, quiet = TRUE)
evalq_clean(expr, quiet = TRUE)

Arguments

expr an R expression to evaluate. For eval_clean this should already be quoted. For evalq_clean it will be quoted for you.

quiet if TRUE, the default, only the final result and the any explicitly printed output will be displayed. If FALSE, all input and output will be displayed, as if you’d copied and paste the code.
**github_pull**

**Value**

An invisible TRUE on success.

**Examples**

```r
x <- 1
y <- 2
ls()
evalq_clean(ls())
evalq_clean(ls(), FALSE)
eval_clean(quote({
  z <- 1
  ls()
}))
```

---

**github_pull**  
*GitHub references*

**Description**

Use as ref parameter to `install_github`. Allows installing a specific pull request or the latest release.

**Usage**

```r
github_pull(pull)
github_release()
```

**Arguments**

<table>
<thead>
<tr>
<th>pull</th>
<th>The pull request to install</th>
</tr>
</thead>
</table>

**See Also**

`install_github`
has_devel

Check if you have a development environment installed.

Description

Thanks to the suggestion of Simon Urbanek.

Usage

has_devel()

Value

TRUE if your development environment is correctly set up, otherwise returns an error.

Examples

has_devel()

help

Drop-in replacements for help and ? functions

Description

The ? and help functions are replacements for functions of the same name in the utils package. They are made available when a package is loaded with load_all.

Usage

# help(topic, package = NULL, ...)

# ?e2
# e1?e2

Arguments

topic A name or character string specifying the help topic.
package A name or character string specifying the package in which to search for the help topic. If NULL, search all packages.
... Additional arguments to pass to help.
e1 First argument to pass along to utils::'?'.
e2 Second argument to pass along to utils::'?'.

infrastructure

Details

The `?` function is a replacement for `?` from the utils package. It will search for help in devtools-loaded packages first, then in regular packages.

The `help` function is a replacement for `help` from the utils package. If `package` is not specified, it will search for help in devtools-loaded packages first, then in regular packages. If `package` is specified, then it will search for help in devtools-loaded packages or regular packages, as appropriate.

Examples

```r
## Not run:
# This would load devtools and look at the help for load_all, if currently
# in the devtools source directory.
load_all()
?load_all
help("load_all")

## End(Not run)

# To see the help pages for utils::help and utils::'?':
help("help", "utils")
help("?", "utils")

## Not run:
# Examples demonstrating the multiple ways of supplying arguments
# NB: you can't do pkg <- "ggplot2"; help("ggplot2", pkg)
help(lm)
help(lm, stats)
help(lm, 'stats')
help('lm')
help('lm', stats)
help('lm', 'stats')
help(package = stats)
help(package = 'stats')
topic <- "lm"
help(topic)
help(topic, stats)
help(topic, 'stats')

## End(Not run)
```

infrastructure

Add useful infrastructure to a package.

Description

Add useful infrastructure to a package.
Usage

use_testthat(pkg = ".")

use_test(name, pkg = ".")

use_rstudio(pkg = ".")

use_vignette(name, pkg = ".")

use_rcpp(pkg = ".")

use_travis(pkg = ".", browse = interactive())

use_coverage(pkg = ".", type = c("codecov", "coveralls"))

use_appveyor(pkg = ".")

use_package_doc(pkg = ".")

use_revdep(pkg = ".")

use_cran_comments(pkg = ".")

use_code_of_conduct(pkg = ".")

use_cran_badge(pkg = ".")

use_mit_license(pkg = ".", copyright_holder = getOption("devtools.name", 
  
  "<Author>"))

use_gpl3_license(pkg = ".")

use_dev_version(pkg = ".")

Arguments

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

name File name to use for new vignette. Should consist only of numbers, letters, _, and -. I recommend using lower case.

browse open a browser window to enable Travis builds for the package automatically.

type CI tool to use. Currently supports codecov and coverall.

copyright_holder The copyright holder for this package. Defaults to getOption("devtools.name").
`use_testthat`

Add testing infrastructure to a package that does not already have it. This will create `tests/testthat.R`, `tests/testthat/` and add `testthat` to the suggested packages. This is called automatically from `test` if needed.

`use_test`

Add a test file, also add testing infrastructure if necessary. This will create `tests/testthat/test-<name>.R` with a user-specified name for the test. Will fail if the file exists.

`use_vignette`

Adds needed packages to DESCRIPTION, and creates draft vignette in vignettes/. It adds inst/doc to .gitignore so you don’t accidentally check in the built vignettes.

`use_rcpp`

Creates src/ and adds needed packages to DESCRIPTION.

`use_travis`

Add basic travis template to a package. Also adds .travis.yml to .buildignore so it isn’t included in the built package.

`use_coverage`

Add test code coverage to basic travis template to a package.

`use_appveyor`

Add basic AppVeyor template to a package. Also adds appveyor.yml to .buildignore so it isn’t included in the built package.

`use_package_doc`

Adds a roxygen template for package documentation

`use_revdep`

Add revdep directory and basic check template.

`use_cran_comments`

Add cran-comments.md template.

`use_code_of_conduct`

Add a code of conduct to from http://contributor-covenant.org.
use_cran_badge

Add a badge to show CRAN status and version number on the README

use_mit_license

Add the necessary infrastructure to declare your package as distributed under the MIT license.

use_gpl3_license

Adds the necessary infrastructure to declare your package as distributed under the GPL v3.

use_dev_version

This adds ".9000" to the package DESCRIPTION, adds a new heading to NEWS.md (if it exists), and then checks the result into git.

See Also

Other infrastructure: use_build_ignore, use_data_raw, use_data, use_news_md, use_package, use_readme_rmd

inst

Get the installation path of a package

Description

Given the name of a package, this returns a path to the installed copy of the package, which can be passed to other devtools functions.

Usage

inst(name)

Arguments

name the name of a package.

Details

It searches for the package in .libPaths(). If multiple dirs are found, it will return the first one.

Examples

inst("devtools")
inst("grid")
## Not run:
# Can be passed to other devtools functions
unload(inst("ggplot2"))
## End(Not run)
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

```r
install(pkg = ".", reload = TRUE, quick = FALSE, local = TRUE,
       args = getOption("devtools.install.args"), quiet = FALSE,
       dependencies = NA, upgrade_dependencies = TRUE, build_vignettes = FALSE,
       keep_source = getOption("keep.source.pkgs"),
       threads = getOption("Ncpus", 1), force_deps = FALSE,
       metadata = remote_metadata(as.package(pkg)),
       out_dir = NULL, skip_if_log_exists = 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.

- **local**
  if FALSE builds 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 suppresses output from this function.

- **dependencies**
  logical indicating to also install uninstalled packages which this pkg depends on/links to/suggests. See argument dependencies of `install.packages`.

- **upgrade_dependencies**
  If TRUE, the default, will also update any out of date dependencies.

- **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".

- **threads**
  number of concurrent threads to use for installing dependencies. It defaults to the option "Ncpus" or 1 if unset.
install_bioc

force_deps
metadata
out_dir
skip_if_log_exists
... additional arguments passed to install.packages when installing dependencies. pkg is installed with R CMD INSTALL.

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 local = FALSE 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 with_libpaths.

See Also

with_debug to install packages with debugging flags set.
Other package installation: install_bioc, install_bitbucket, install_cran, install_github, install_git, install_svn, install_url, install_version, uninstall

install_bioc Install a package from a Bioconductor repository

Description

This function requires svn to be installed on your system in order to be used.

Usage

install_bioc(repo, mirror =getOption("BioC_svn",
"https://hedgehog.fhcrc.org/bioconductor"), ..., quiet = FALSE)

Arguments

description

repo
mirror
... other arguments passed to install
quiet

**install_bitbucket**

**Details**

It is vectorised so you can install multiple packages with a single command.

**See Also**

Other package installation: `install_bitbucket`, `install_cran`, `install_github`, `install_git`, `install_svn`, `install_url`, `install_version`, `install`, `uninstall`

**Examples**

```r
## Not run:
install_bioc("SummarizedExperiment")
install_bioc("user@SummarizedExperiment")
install_bioc("user:password@release/SummarizedExperiment")
install_bioc("user:password@0.3/SummarizedExperiment")
install_bioc("user:password@0.3/SummarizedExperiment#117513")
```

```r
## End(Not run)
```

---

**install_bitbucket**  
*Install a package directly from bitbucket*

**Description**

This function is vectorised so you can install multiple packages in a single command.

**Usage**

```r
install_bitbucket(repo, username, ref = "master", subdir = NULL, quiet = FALSE, auth_user = NULL, password = NULL, ...)
```

**Arguments**

- **repo**: Repository address in the format `username/repo[/subdir][@ref|#pull]`. Alternatively, you can specify `subdir` and/or `ref` using the respective parameters (see below); if both are specified, the values in `repo` take precedence.

- **username**: User name. Deprecated: please include username in the `repo`.

- **ref**: Desired git reference; could be a commit, tag, or branch name. Defaults to `master`.

- **subdir**: Subdirectory within `repo` that contains the R package.

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

- **auth_user**: Your account username if you’re attempting to install a package hosted in a private repository (and your username is different to `username`).

- **password**: Your password.

- **...**: Other arguments passed on to `install`. 
install_cran

See Also

Bitbucket API docs: https://confluence.atlassian.com/bitbucket/use-the-bitbucket-cloud-rest-apis-22272.html

Other package installation: install_bioc, install_cran, install_github, install_git, install_svn, install_url, install_version, install, uninstall

Examples

## Not run:
install_bitbucket("sulab/mygene.r@default")
install_bitbucket("dannavarro/lsr-package")

## End(Not run)

install_cran

Attempts to install a package from CRAN.

Description

This function is vectorised on pkgs so you can install multiple packages in a single command.

Usage

install_cran(pkgs, repos =getOption("repos"), type =getOption("pkgType"),
... , quiet = FALSE)

Arguments

pkgs Character vector of packages to install.
repos A character vector giving repositories to use.
type Type of package to update. If "both", will switch automatically to "binary" to avoid interactive prompts during package installation.
... Additional arguments passed to install_packages.
quiet If TRUE, suppress output.

See Also

Other package installation: install_bioc, install_bitbucket, install_github, install_git, install_svn, install_url, install_version, install, uninstall

Examples

## Not run:
install_cran("ggplot2")
install_cran(c("httpuv", "shiny")

## End(Not run)
install_deps

Install package dependencies if needed.

Description

install_deps is used by install_* to make sure you have all the dependencies for a package. install_dev_deps() is useful if you have a source version of the package and want to be able to develop with it: it installs all dependencies of the package, and it also installs roxygen2.

Usage

install_deps(pkg = ".", dependencies = NA, threads = getOption("Ncpus", 1), repos = getOption("repos"), type = getOption("pkgType"), ..., upgrade = TRUE, quiet = FALSE, force_deps = FALSE)

install_dev_deps(pkg = ".", ...)

Arguments

pkg package description, can be path or package name. See as.package for more information.
dependencies logical indicating to also install uninstalled packages which this pkg depends on/links to/suggests. See argument dependencies of install.packages.
threads number of concurrent threads to use for installing dependencies. It defaults to the option "Ncpus" or 1 if unset.
repos A character vector giving repositories to use.
type Type of package to update. If "both", will switch automatically to "binary" to avoid interactive prompts during package installation.
... additional arguments passed to install.packages.
upgrade If TRUE, also upgrade any of out date dependencies.
quiet if TRUE suppresses output from this function.
force_deps whether to force installation of dependencies even if their SHA1 reference hasn’t changed from the currently installed version.

Examples

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

Install a package from a git repository

Description

It is vectorised so you can install multiple packages with a single command. You do not need to have git installed.

Usage

install_git(url, subdir = NULL, branch = NULL, credentials = NULL, quiet = FALSE, ...)

Arguments

- `url` Location of package. The url should point to a public or private repository.
- `subdir` A sub-directory within a git repository that may contain the package we are interested in installing.
- `branch` Name of branch or tag to use, if not master.
- `credentials` A git2r credentials object passed through to `clone`.
- `quiet` if TRUE suppresses output from this function.
- `...` passed on to `install`

See Also

Other package installation: `install_bioc`, `install_bitbucket`, `install_cran`, `install_github`, `install_svn`, `install_url`, `install_version`, `install`, `uninstall`

Examples

```r
## Not run:
install_git("git://github.com/hadley/stringr.git")
install_git("git://github.com/hadley/stringr.git", branch = "stringr-0.2")
```

## End(Not run)
**install_github**  
Attempts to install a package directly from GitHub.

Description

This function is vectorised on repo so you can install multiple packages in a single command.

Usage

```r
install_github(repo, username = NULL, ref = "master", subdir = NULL,
               auth_token = github_pat(quiet), host = "https://api.github.com",
               quiet = FALSE, ...)
```

Arguments

- `repo`  
  Repository address in the format `username/repo[@subdir][@ref|#pull]`. Alternatively, you can specify `subdir` and/or `ref` using the respective parameters (see below); if both are specified, the values in `repo` take precedence.

- `username`  
  User name. Deprecated: please include username in the `repo`.

- `ref`  
  Desired git reference. Could be a commit, tag, or branch name, or a call to `github_pull`. Defaults to "master".

- `subdir`  
  Subdirectory within repo that contains the R package.

- `auth_token`  
  To install from a private repo, generate a personal access token (PAT) in [https://github.com/settings/tokens](https://github.com/settings/tokens) and supply to this argument. This is safer than using a password because you can easily delete a PAT without affecting any others. Defaults to the `GITHUB_PAT` environment variable.

- `host`  
  GitHub API host to use. Override with your GitHub enterprise hostname, for example, "github.hostname.com/api/v3".

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

- `...`  
  Other arguments passed on to `install`.

Details

Attempting to install from a source repository that uses submodules raises a warning. Because the zipped sources provided by GitHub do not include submodules, this may lead to unexpected behaviour or compilation failure in source packages. In this case, cloning the repository manually using `install_git` with `args="--recursive"` may yield better results.

See Also

- `github_pull`

Other package installation: `install_bioc`, `install_bitbucket`, `install_cran`, `install_git`, `install_svn`, `install_url`, `install_version`, `install`, `uninstall`
install_local

install_local
Install a package from a local file

Description
This function is vectorised so you can install multiple packages in a single command.

Usage
install_local(path, subdir = NULL, ..., quiet = FALSE)

Arguments
path path to local directory, or compressed file (tar, zip, tar.gz tar.bz2, tgz2 or tbz)
subdir subdirectory within url bundle that contains the R package.
... Other arguments passed on to install.
quiet if TRUE suppresses output from this function.

Examples
## Not run:
dir <- tempfile()
dir.create(dir)
pkg <- download.packages("testthat", dir, type = "source")
install_local(pkg[, 2])

## End(Not run)
install_svn

Install a package from a SVN repository

Description

This function requires svn to be installed on your system in order to be used.

Usage

install_svn(url, subdir = NULL, branch = NULL, args = character(0), ..., revision = NULL, quiet = FALSE)

Arguments

url Location of package. The url should point to a public or private repository.
subdir A sub-directory within asvn repository that may contain the package we are interested in installing. By default, this points to the 'trunk' directory.
branch Name of branch or tag to use, if not trunk.
args A character vector providing extra arguments to pass on to
... Other arguments passed on to install
revision svn revision, if omitted updates to latest
quiet if TRUE suppresses output from this function.

Details

It is vectorised so you can install multiple packages with a single command.

See Also

Other package installation: install_bioc, install_bitbucket, install_cran, install_github, install_git, install_url, install_version, install, uninstall

Examples

```r
## Not run:
install_svn("https://github.com/hadley/stringr")
install_svn("https://github.com/hadley/httr", branch = "oauth")
## End(Not run)
```
install_url

Install a package from a url

Description

This function is vectorised so you can install multiple packages in a single command.

Usage

install_url(url, subdir = NULL, config = list(), ..., quiet = FALSE)

Arguments

url
location of package on internet. The url should point to a zip file, a tar file or a bzipped/gzipped tar file.

subdir
subdirectory within url bundle that contains the R package.

config
additional configuration argument (e.g. proxy, authentication) passed on to GET.

...
Other arguments passed on to install.

quiet
if TRUE suppresses output from this function.

See Also

Other package installation: install_bioc, install_bitbucket, install_cran, install_github, install_git, install_svn, install_version, install, uninstall

Examples

## Not run:
install_url("https://github.com/hadley/stringr/archive/master.zip")

## End(Not run)

install_version

Install specified version of a CRAN package.

Description

If you are installing an package that contains compiled code, you will need to have an R development environment installed. You can check if you do by running has_devel.

Usage

install_version(package, version = NULL, repos =getOption("repos"),
    type = getOption("pkgType"), ..., quiet = FALSE)
Arguments

package: package name

version: If the specified version is NULL or the same as the most recent version of the package, this function simply calls `install`. Otherwise, it looks at the list of archived source tarballs and tries to install an older version instead.

repos: character vector, the base URL(s) of the repositories to use, e.g., the URL of a CRAN mirror such as "https://cloud.r-project.org". For more details on supported URL schemes see `url`. Can be NULL to install from local files, directories or URLs: this will be inferred by extension from `pkgs` if of length one.

type: character, indicating the type of package to download and install. Will be "source" except on Windows and some macOS builds: see the section on ‘Binary packages’ for those.

...: Other arguments passed on to `install`.

quiet: logical: if true, reduce the amount of output.

Author(s)

Jeremy Stephens

See Also

Other package installation: `install_bioc`, `install_bitbucket`, `install_cran`, `install_github`, `install_git`, `install_svn`, `install_url`, `install`, `uninstall`

lint

Lint all source files in a package.

Description

The default lintings 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 `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 `clear_cache`.

See Also

`lint_package`, `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(pkg = ", reset = TRUE, recompile = FALSE, export_all = TRUE, quiet = FALSE, create = NA)`

Arguments

- **pkg**: package description, can be path or package name. See `as.package` for more information.
- **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 `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.
- **quiet**: if TRUE suppresses output from this function.
- **create**: only relevant if a package structure does not exist yet: if TRUE, create a package structure; if NA, ask the user (in interactive mode only)

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.
load_all

• 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.

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 laded 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

```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
# are exported
load_all("./", export_all = FALSE)

## End(Not run)
```
load_code

Load R code.

Description
Load all R code in the R directory. The first time the code is loaded, .onLoad will be run if it exists.

Usage
load_code(pkg = ".")

Arguments
pkg package description, can be path or package name. See `as.package` for more information

load_data

Load data.

Description
Loads all .RData files in the data subdirectory.

Usage
load_data(pkg = ".")

Arguments
pkg package description, can be path or package name. See `as.package` for more information

load_dll

Load a compiled DLL

Description
Load a compiled DLL

Usage
load_dll(pkg = ".")

Arguments
pkg package description, can be path or package name. See `as.package` for more information
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_deps  

**Find all dependencies of a CRAN or dev package.**

**Description**
Find all the dependencies of a package and determine whether they are ahead or behind CRAN. A print() method identifies mismatches (if any) between local and CRAN versions of each dependent package; an update() method installs outdated or missing packages from CRAN.

**Usage**

```r
package_deps(pkg, dependencies = NA, repos = getOption("repos"),
              type = getOption("pkgType"))
```

```r
dev_package_deps(pkg = ".", dependencies = NA, repos = getOption("repos"),
                 type = getOption("pkgType"), bioconductor = TRUE)
```

```r
## S3 method for class 'package_deps'
update(object, ..., quiet = FALSE, upgrade = TRUE)
```

**Arguments**

- **pkg**  
  A character vector of package names. If missing, defaults to the name of the package in the current directory.

- **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.

type: Type of package to update. If "both", will switch automatically to "binary" to avoid interactive prompts during package installation.

bioconductor: Install Bioconductor dependencies if the package has a BiocViews field in the DESCRIPTION.

object: A package_deps object.

...: Additional arguments passed to install_packages.

quiet: If TRUE, suppress output.

upgrade: If TRUE, also upgrade any out of date dependencies.

Value

A data.frame with columns:

- package: The dependent package's name.
- installed: The currently installed version.
- available: The version available on CRAN.
- diff: An integer denoting whether the locally installed version of the package is newer (1), the same (0) or older (-1) than the version currently available on CRAN.

Examples

```r
## Not run:
package_deps("devtools")
# Use update to update any out-of-date dependencies
update(package_deps("devtools"))

## End(Not run)
```

---

package_file: Find file in a package.

Description

It always starts by finding 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.
path

Examples

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

## End(Not run)
```

---

*path*  
*Get/set the PATH variable.*

---

**Description**

Get/set the PATH variable.

**Usage**

```r
get_path()

set_path(path)

add_path(path, after = Inf)
```

**Arguments**

- `path` character vector of paths
- `after` for `add_path`, the place on the PATH where the new paths should be added

**Value**

`set_path` invisibly returns the old path.

**See Also**

*with_path* to temporarily set the path for a block of code

Other path: *on_path*

**Examples**

```r
path <- get_path()
length(path)
old <- add_path(".")
length(get_path())
set_path(old)
length(get_path())
```
RCMD

Run R CMD xxx from within R

Description

Run R CMD xxx from within R

Usage

RCMD(cmd, options, path = tempdir(), env_vars = character(), ...)

Arguments

cmd
one of the R tools available from the R CMD interface.

options
a character vector of options to pass to the command

path
the directory to run the command in.

env_vars
environment variables to set before running the command.

... additional arguments passed to system_check

Value

TRUE if the command succeeds, throws an error if the command fails.

release

Release package to CRAN.

Description

Run automated and manual tests, then ftp to CRAN.

Usage

release(pkg = ".", check = TRUE, args = NULL, spelling = "en_US")

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.

spelling
language or dictionary file to spell check documentation. See spell_check. Set to NULL to skip spell checking.
Details

The package release process will:

- Confirm that the package passes `R CMD check`
- Ask if you’ve checked your code on win-builder
- Confirm that news is up-to-date
- Confirm that DESCRIPTION is ok
- Ask if you’ve checked packages that depend on your package
- Build the package
- Submit the package to CRAN, using comments in "cran-comments.md"

You can also 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.

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.

---

**reload**

*Unload and reload package.*

---

**Description**

This attempts to unload and reload a 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** package description, can be path or package name. See `as.package` for more information
- **quiet** if TRUE suppresses output from this function.
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(inst("ggplot2"))

## End(Not run)
```

revdep

Reverse dependency tools.

Description

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

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

## Not run:
revdep("ggplot2")

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

## End(Not run)

revdep_check_save_summary

Run R CMD check on all downstream dependencies.

Description

Use revdep_check() to run check_cran() on all downstream dependencies. Summarises the results with revdep_check_summary() and see problems with revdep_check_print_problems().

Usage

revdep_check_save_summary(pkg = ".")

revdep_check_print_problems(pkg = ".")

revdep_check(pkg = ".", recursive = FALSE, ignore = NULL,
dependencies = c("Depends", "Imports", "Suggests", "LinkingTo"),
skip = character(), libpath = getOption("devtools.revdep.libpath"),
srcpath = libpath, bioconductor = FALSE, type = getOption("pkgType"),
threads = getOption("Ncpus", 1), env_vars = NULL, check_dir = NULL,
install_dir = NULL, quiet_check = TRUE)

revdep_check_resume(pkg = ".", ...)

revdep_check_reset(pkg = ".")

Arguments

pkg Path to package. Defaults to current directory.

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.

dependencies A character vector listing the types of dependencies to follow.
skip
libpath
srcpath
bioconductor
type
threads
env_vars
check_dir
install_dir
quiet_check

Details

Revdep checks are resumable - this is very helpful if somethings goes wrong (like you run out of power or you lose your internet connection) in the middle of a check. You can resume a partially completed check with revdep_check_resume(), or blow away the cached result so you can start afresh with revdep_check_reset().

Value

An invisible list of results. But you’ll probably want to look at the check results on disk, which are saved in check_dir. Summaries of all ERRORs and WARNINGs will be stored in check_dir/0@check-summary.txt.

Check process

1. Install pkg (in special library, see below).
2. Find all CRAN packages that depend on pkg.
3. Install those packages, along with their dependencies.
4. Run R CMD check on each package.
5. Uninstall pkg (so other reverse dependency checks don’t use the development version instead of the CRAN version)
Package library

By default revdep_check uses a temporary library to store any packages that are required by the packages being tested. This ensures that they don’t interfere with your default library, but means that if you restart R between checks, you’ll need to reinstall all the packages. If you’re doing reverse dependency checks frequently, I recommend that you create a directory for these packages and set `options(devtools.revdep.libpath)`.

See Also

revdep_maintainers() to get a list of all revdep maintainers.

Examples

```r
## Not run:
# Run R CMD check on all downstream dependencies
revdep_check()
revdep_check_save_summary()
revdep_check_print_problems()

## End(Not run)
```

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)
```

Arguments

- `pkg` package description, can be path or package name. 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` if TRUE, code in `\dontshow{}` will be commented out.
- `test` if TRUE, code in `\donttest{}` will be commented out. If FALSE, code in `\testonly{}` will be commented out.
run if TRUE, 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.

See Also

Other example functions: `dev_example`

---

**session_info**

Print session information

**Description**

This is `sessionInfo()` re-written from scratch to both exclude data that's rarely useful (e.g., the full collate string or base packages loaded) and include stuff you'd like to know (e.g., where a package was installed from).

**Usage**

`session_info(pkgs = NULL, include_base = FALSE)`

**Arguments**

pkgs Either a vector of package names or NULL. If NULL, displays all loaded packages. If a character vector, also, includes all dependencies of the package.

include_base Include base packages in summary? By default this is false since base packages should always match the R version.

**Examples**

`session_info()`

`session_info("devtools")`

---

**show_news**

Show package news

**Description**

Show package news

**Usage**

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

Run a script on gist

**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

---

**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/](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.

**Examples**

```r
## 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.

### 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 = "54f1db27e60bb7e0486d7856049e49e8fe9f9")

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

## End(Not run)
```
spell_check

Description

Runs a spell check on text fields in the package description file and manual pages. Hunspell includes dictionaries for en_US and en_GB by default. Other languages require installation of a custom dictionary, see the hunspell vignette for details.

Usage

spell_check(pkg = ".", ignore = character(), dict = "en_US")

Arguments

pkg package description, can be path or package name. See as.package for more information
ignore character vector with words to ignore. See hunspell for more information
dict a dictionary object or language string. See hunspell for more information

system.file

Replacement version of system.file

Description

This function is meant to intercept calls to system.file, so that it behaves well with packages loaded by devtools. It is made available when a package is loaded with load_all.

Usage

# system.file(., package = "base", lib.loc = NULL, mustWork = FALSE)

Arguments

... character vectors, specifying subdirectory and file(s) within some package. The default, none, returns the root of the package. Wildcards are not supported.
package a character string with the name of a single package. An error occurs if more than one package name is given.
lib.loc a character vector with path names of R libraries. See ‘Details’ for the meaning of the default value of NULL.
mustWork logical. If TRUE, an error is given if there are no matching files.
Details

When `system.file` is called from the R console (the global environment), this function detects if the target package was loaded with `load_all`, and if so, it uses a customized method of searching for the file. This is necessary because the directory structure of a source package is different from the directory structure of an installed package.

When a package is loaded with `load_all`, this function is also inserted into the package’s imports environment, so that calls to `system.file` from within the package namespace will use this modified version. If this function were not inserted into the imports environment, then the package would end up calling `base::system.file` instead.

---

**test**  
*Execute all test_that tests in a package.*

---

Description

Tests are assumed to be located in either the `inst/tests/` or `tests/testthat` directory (the latter is recommended). See `test_dir` for the naming convention of test scripts within one of those directories and `test_check` for the folder structure conventions.

Usage

```
test(pkg = ".", filter = NULL, ...)
uses_testthat(pkg = ".")
```

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 `test_dir`

Details

If no testing infrastructure is present (detected by the `uses_testthat` function), you’ll be asked if you want devtools to create it for you (in interactive sessions only). See `use_test` for more details.
uninstall | Uninstall a local development package.

**Description**

Uses `remove.package` to uninstall the package. To uninstall a package from a non-default library, use `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` suppresses output from this function.
- `...`: additional arguments passed to `remove.packages`.

**See Also**

- `with_debug` to install packages with debugging flags set.
- Other package installation: `install.bioc, install.bitbucket, install.cran, install.github, install.git, install.svn, install.url, install.version, install`.

unload | Unload a package

**Description**

This function attempts to cleanly unload a package, including unloading its namespace, deleting S4 class definitions and unloading any loaded DLLs. Unfortunately S4 classes are not really designed to be cleanly unloaded, and so we have to manually modify the class dependency graph in order for it to work - this works on the cases for which we have tested but there may be others. Similarly, automated DLL unloading is best tested for simple scenarios (particularly with `useDynLib(pkgname)`) and may fail in other cases. If you do encounter a failure, please file a bug report at [http://github.com/r-lib/devtools/issues](http://github.com/r-lib/devtools/issues).

**Usage**

```r
unload(pkg = ".")
```
**update_packages**

Update packages that are missing or out-of-date.

**Description**

Works similarly to `install.packages()` but doesn’t install packages that are already installed, and also upgrades out dated dependencies.

**Usage**

```r
update_packages(pkgs = NULL, dependencies = NA, repos = getOption("repos"), type = getOption("pkgType"))
```

**Arguments**

- **pkgs**
  - Character vector of packages to update. If `TRUE` all installed packages are updated. If `NULL` user is prompted to confirm update of all installed packages.

- **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.

- **repos**
  - A character vector giving repositories to use.

- **type**
  - Type of package to update. If "both", will switch automatically to "binary" to avoid interactive prompts during package installation.
use_data

See Also

package_deps to see which packages are out of date/ missing.

Examples

```r
## Not run:
update_packages("ggplot2")
update_packages(c("plyr", "ggplot2"))
```

## End(Not run)

use_data

Use data in a package.

Description

This function makes it easy to save package data in the correct format.

Usage

```r
use_data(..., pkg = ".", internal = FALSE, overwrite = FALSE, 
compress = "bzip2")
```

Arguments

- `...` Unquoted names of existing objects to save.
- `pkg` Package where to store data. Defaults to package in working directory.
- `internal` If FALSE, saves each object in individual .rda files in the data/ directory. These are available whenever the package is loaded. If TRUE, stores all objects in a single R/sysdata.rda file. These objects are only available within the package.
- `overwrite` By default, use_data will not overwrite existing files. If you really want to do so, set this to TRUE.
- `compress` Choose the type of compression used by save. Should be one of "gzip", "bzip2" or "xz".

See Also

Other infrastructure: infrastructure, use_build_ignore, use_data_raw, use_news_md, use_package, use_readme_rmd
Examples

```r
## Not run:
x <- 1:10
y <- 1:100

use_data(x, y) # For external use
use_data(x, y, internal = TRUE) # For internal use

## End(Not run)
```

use_data_raw

Use data-raw to compute package datasets.

Description

Use data-raw to compute package datasets.

Usage

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

Arguments

- `pkg` Package where to create data-raw. Defaults to package in working directory.

See Also

Other infrastructure: `infrastructure, use_build_ignore, use_data, use_news_md, use_package, use_readme_rmd`

use_git

Initialise a git repository.

Description

Initialise a git repository.

Usage

```r
use_git(message = "Initial commit", pkg = ".")
```

Arguments

- `message` Message to use for first commit.
- `pkg` Path to package. See `as.package` for more information.
**use_github**

*Connect a local repo with GitHub.*

## Description

If the current repo does not use git, calls `use_git` automatically. `use_github_links` is called to populate the URL and BugReports fields of DESCRIPTION.

## Usage

```r
use_github(auth_token = github_pat(), private = FALSE, pkg = ".", 
            host = "https://api.github.com", protocol = c("ssh", "https"),
            credentials = NULL)
```

## Arguments

- **auth_token**: Provide a personal access token (PAT) from [https://github.com/settings/tokens](https://github.com/settings/tokens). Defaults to the GITHUB_PAT environment variable.
- **private**: If TRUE, creates a private repository.
- **pkg**: Path to package. See `as.package` for more information.
- **host**: GitHub API host to use. Override with the endpoint-root for your GitHub enterprise instance, for example, "https://github.hostname.com/api/v3".
- **protocol**: Transfer protocol, either "ssh" (the default) or "https"
- **credentials**: A `cred_ssh_key` specifying specific ssh credentials or NULL for default ssh key and ssh-agent behaviour. Default is NULL.

## Authentication

A new GitHub repo will be created via the GitHub API, therefore you must provide a GitHub personal access token (PAT) via the `auth_token` argument, which defaults to the value of the GITHUB_PAT environment variable. Obtain a PAT from [https://github.com/settings/tokens](https://github.com/settings/tokens).

The "repo" scope is required which is one of the default scopes for a new PAT.

The argument protocol reflects how you wish to authenticate with GitHub for this repo in the long run. For either protocol, a remote named "origin" is created, an initial push is made using the specified protocol, and a remote tracking branch is set. The URL of the "origin" remote has the form `git@github.com:<USERNAME>/<REPO>.git` (protocol = "ssh", the default) or `https://github.com/<USERNAME>/<REPO>.git` (protocol = "https"). For protocol = "ssh", it is assumed that public and private keys are in the default locations, `.ssh/id_rsa.pub` and `~/.ssh/id_rsa`, respectively, and that ssh-agent is configured to manage any associated passphrase. Alternatively, specify a `cred_ssh_key` object via the `credentials` parameter.
See Also

Other git infrastructure: use_git_hook, use_github_links, use_git

Examples

```r
## Not run:
## to use default ssh protocol
create("testpkg")
use_github(pkg = "testpkg")

## or use https
create("testpkg2")
use_github(pkg = "testpkg2", protocol = "https")

## End(Not run)
```

---

**use_news_md**

*Use NEWS.md*

Description

This creates NEWS.md from a template.

Usage

```
use_news_md(pkg = ".")
```

Arguments

pkg

package description, can be path or package name. See *as.package* for more information

See Also

Other infrastructure: infrastructure, use_build_ignore, use_data_raw, use_data, use_package, use_readme_rmd
**use_package**  
Use specified package.

**Description**  
This adds a dependency to DESCRIPTION and offers a little advice about how to best use it.

**Usage**  
```
use_package(package, type = "Imports", pkg = ".")
```

**Arguments**  
- **package**: Name of package to depend on.
- **type**: Type of dependency: must be one of "Imports", "Depends", "Suggests", "Enhances", or "LinkingTo" (or unique abbreviation)
- **pkg**: package description, can be path or package name. See `as.package` for more information.

**See Also**  
Other infrastructure: `infrastructure`, `use_build_ignore`, `use_data_raw`, `use_data`, `use_news_md`, `use_readme_rmd`

**Examples**  
```r
## Not run:
use_package("ggplot2")
use_package("dplyr", "suggests")

## End(Not run)
```

---

**use_readme_rmd**  
Create README files.

**Description**  
Creates skeleton README files with sections for
- a high-level description of the package and its goals
- R code to install from GitHub, if GitHub usage detected
- a basic example

Use `Rmd` if you want a rich intermingling of code and data. Use `md` for a basic README. README.Rmd will be automatically added to .Rbuildignore. The resulting README is populated with default YAML frontmatter and R fenced code blocks (md) or chunks (Rmd).
Usage

```r
use_readme_rmd(pkg = ".")
use_readme_md(pkg = ".")
```

Arguments

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

See Also

Other infrastructure: `infrastructure`, `use_build_ignore`, `use_data_raw`, `use_data`, `use_news_md`, `use_package`

Examples

```r
## Not run:
use_readme_rmd()
use_readme_md()

## End(Not run)
```

---

`wd` *Set working directory.*

Description

Set working directory.

Usage

```r
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.
with_debug

Temporarily set debugging compilation flags.

Description

Temporarily set debugging compilation flags.

Usage

with_debug(code, CFLAGS = NULL, CXXFLAGS = NULL, FFLAGS = NULL, FCFLAGS = NULL, debug = TRUE)

Arguments

code        to execute.
CFLAGS      flags for compiling C code
CXXFLAGS    flags for compiling C++ code
FFLAGS      flags for compiling Fortran code.
FCFLAGS     flags for Fortran 9x code.
debug       If TRUE adds -g -00 to all flags (Adding FFLAGS and FCFLAGS

See Also

Other debugging flags: compiler_flags

Examples

flags <- names(compiler_flags(TRUE))
with_debug(Sys.getenv(flags))

## Not run:
install("mypkg")
with_debug(install("mypkg"))

## End(Not run)
Index

*Topic programming
  build_vignettes, 6
  load_all, 36
  load_code, 38
  load_data, 38
  load_dll, 38
  run_examples, 47
.libPaths, 16, 24
?, 21
? (help), 20

add_path (path), 41
add_travis (infrastructure), 21
as.package, 3, 4, 6, 7, 9–12, 17, 22, 25, 29,
  35, 36, 38, 39, 42, 43, 47, 49, 51–54,
  56–60
as.person, 14

bash, 3
browser, 48
build, 4, 7, 25
build_github_devtools, 5
build_vignettes, 6
build_win, 4, 6

check, 7, 9, 13
check_build (check), 7
check_cran, 45
check_failures, 9
check_man, 9
clean_dll, 10, 12, 36
clean_source, 10
clean_vignettes, 6, 11
clear_cache, 36
clone, 30
compile_dll, 10, 11, 12, 37
compiler_flags, 8, 11, 12, 61
create, 12
create_description, 13
cred_ssh_key, 57

dev_example, 15, 48
dev_help, 15
dev_mode, 14, 16
dev_package_deps (package_deps), 39
devtools, 14
devtools-package (devtools), 14
document, 17
dr_devtools, 17, 18
dr_github, 17, 18
eval_clean, 18
evalq_clean (eval_clean), 18
GET, 34
get_path (path), 41
github_pull, 19, 31
github_release (github_pull), 19

has_devel, 20, 34
help, 20, 20, 21
hunspell, 51

infrastructure, 21, 55, 56, 58–60
inst, 24
install, 14, 25, 26–28, 30–35, 53
install.packages, 6, 25, 26, 29, 46
install_bioc, 26, 26, 28, 30, 31, 33–35, 53
install_bitbucket, 26, 27, 27, 28, 30, 31,
  33–35, 53
install_cran, 26–28, 28, 30, 31, 33–35, 53
install_deps, 29
install_dev_deps (install_deps), 29
install_git, 26–28, 30, 31, 33–35, 53
install_github, 5, 19, 26–28, 30, 31, 33–35,
  53
install_local, 32
install_svn, 26–28, 30, 31, 33, 34, 35, 53
install_url, 26–28, 30, 31, 33, 34, 35, 53
install_version, 26–28, 30, 31, 33, 34, 34, 53
library, 36, 37, 43
lint, 35, 36
lint_package, 35, 36
load_all, 20, 36, 51, 52
load_code, 36, 38
load_data, 36, 38
load_dll, 38
missing_s3, 39
on_path, 41
options, 14
package.skeleton, 12, 13
package_deps, 39, 55
package_file, 40
path, 41
r_env_vars, 8
RCMD, 42
release, 8, 42
reload, 26, 43
remove.packages, 53
revdep, 44
revdep_check, 9, 44, 45
revdep_check
  (revdep_check_save_summary), 45
revdep_check_print_problems
  (revdep_check_save_summary), 45
revdep_check_reset
  (revdep_check_save_summary), 45
revdep_check_resume
  (revdep_check_save_summary), 45
revdep_check_save_summary, 45
revdep_maintainers, 47
revdep_maintainers (revdep), 44
roxygenize, 17
run_examples, 15, 47
save, 55
session_info, 48
sessionInfo, 48
set_path (path), 41
setup (create), 12
shim_help (help), 20
shim_question (help), 20
shim_system_file (system_file), 51
show_news, 48
source, 49, 50
source_gist, 49
source_url, 49, 50
spell_check, 49, 50
system_file, 51, 51
system_check, 42
test, 23, 52
test_check, 52
test_dir, 52
uninstall, 26–28, 30, 31, 33–35, 53
unload, 36, 43, 53
update.package_deps (package_deps), 39
update_packages, 54
url, 35
use_appveyor (infrastructure), 21
use_build_ignore, 24, 55, 56, 58–60
use_code_of_conduct (infrastructure), 21
use_coverage (infrastructure), 21
use_cran_badge (infrastructure), 21
use_cran_comments (infrastructure), 21
use_data, 24, 55, 56, 58–60
use_data_raw, 24, 55, 56, 58–60
use_dev_version (infrastructure), 21
use_git, 56, 57, 58
use_git_hook, 57, 58
use_github, 57, 57
use_github_links, 57, 58
use_gpl3_license (infrastructure), 21
use_mit_license (infrastructure), 21
use_news_md, 24, 55, 56, 58, 59, 60
use_package, 24, 55, 56, 58, 59, 60
use_package_doc (infrastructure), 21
use_rcpp (infrastructure), 21
use_readme_md (use_readme_rmd), 59
use_readme_rmd, 24, 55, 56, 58, 59, 59
use_revdep (infrastructure), 21
use_rstudio, 13
use_rstudio (infrastructure), 21
use_test, 52
use_test (infrastructure), 21
use_testthat (infrastructure), 21
use_travis (infrastructure), 21
use_vignette (infrastructure), 21
uses_testthat (test), 52
wd, 60
with_debug, 11, 26, 53, 61
with_libpaths, 26, 53
with_path, 41