Package ‘devtools’

February 19, 2015

Title Tools to Make Developing R Packages Easier
Version 1.7.0
Description Collection of package development tools.
URL http://github.com/hadley/devtools
BugReports http://github.com/hadley/devtools/issues
Depends R (>= 3.0.2)
Imports httr (>= 0.4), RCurl, utils, tools, methods, memoise, whisker,
evaluate, digest, rstudioapi (>= 0.2.0), jsonlite, roxygen2 (>= 4.1.0)
Suggests testthat (>= 0.7), BiocInstaller, Rcpp (>= 0.10.0), MASS,
rmarkdown, knitr, lintr
License GPL (>= 2)
Author Hadley Wickham [aut, cre],
Winston Chang [aut],
RStudio [cph],
R Core team [ctb] (Some namespace and vignette code extracted from base
R)
Maintainer Hadley Wickham <hadley@rstudio.com>
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-01-17 11:48:37

R topics documented:

bash ........................................... 3
build ......................................... 3
build_github_devtools ....................... 4
build_vignettes ............................. 5
build_win .................................... 6
check ....................................... 6
check_doc .................................. 8
R topics documented:

clean_dll ......................................................... 8
clean_source ...................................................... 9
clean_vignettes ................................................... 9
compiler_flags ..................................................... 10
compile_dll ....................................................... 10
create ............................................................ 11
create_description .............................................. 12
development ....................................................... 13
de_example ......................................................... 13
de_help ............................................................ 14
de_mode ............................................................ 15
document .......................................................... 15
eval_clean ......................................................... 16
github_pull ......................................................... 17
has_devel ......................................................... 17
help ............................................................... 18
infrastructure ..................................................... 19
inst ............................................................... 21
install ............................................................ 21
install_bitbucket .................................................. 23
install_deps ....................................................... 24
install_git ........................................................ 24
install_github ..................................................... 25
install_gitorious .................................................. 26
install_local ...................................................... 27
install_svn ........................................................ 27
install_url ......................................................... 28
install_version ................................................... 29
lint ............................................................... 30
load_all .......................................................... 30
load_code ........................................................ 32
load_data ........................................................ 32
load_dll .......................................................... 33
missing_s3 ........................................................ 33
path .............................................................. 34
release ............................................................ 35
reload ............................................................. 36
revdep ............................................................. 36
revdep_check_save_logs ......................................... 37
run_examples ..................................................... 39
session_info ...................................................... 40
show_news ........................................................ 40
source_gist ....................................................... 41
source_url ........................................................ 42
system.file ....................................................... 43
test .............................................................. 43
unload ............................................................ 44
use_data .......................................................... 45
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

---

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-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

Description

This function is especially useful for Windows users who want to upgrade their version of devtools to the development version hosted on 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).
**build_vignettes**

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 [http://cran.r-project.org/bin/windows/Rtools/](http://cran.r-project.org/bin/windows/Rtools/).

Value

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

Examples

```r
## 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 make files (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

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

Arguments

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

See Also

- `clean_vignettes` to remove the pdfs in `inst/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. 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: these typically represent serious problems.
Usage

    check(pkg = ".", document = TRUE, cleanup = TRUE, cran = TRUE,
    check_version = FALSE, force_suggests = TRUE, args = NULL,
    build_args = NULL, quiet = FALSE, check_dir = tempdir(), ...)

Arguments

pkg          package description, can be path or package name. See as.package for more information
document     if TRUE (the default), will update and check documentation before running formal check.
cleanup      if TRUE the check directory is removed if the check is successful - this allows you to inspect the results to figure out what went wrong. If FALSE the check directory is never removed.
cran         if TRUE (the default), check using the same settings as CRAN uses.
check_version if TRUE, check that the new version is greater than the current version on CRAN, by setting the _R_CHECK_CRAN_INCOMING_ environment variable to TRUE.
force_suggests if FALSE, don’t force suggested packages, by setting the _R_CHECK_FORCE_SUGGESTS_ environment variable to FALSE.
args, build_args An optional character vector of additional command line arguments to be passed to R CMD check/R CMD build/R CMD INSTALL.
quiet        if TRUE suppresses output from this function.
check_dir    the directory in which the package is checked
...          Additional arguments passed to build

Details

    check automatically builds a package before using R CMD check 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.

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).
- Special environment variables set to the same values that CRAN uses when testing packages: cran_env_vars. Unfortunately exactly what CRAN does when checking a package is not publicly documented, but we do our best to simulate as accurately as possible given what we know.
See Also

\texttt{release} if you want to send the checked package to CRAN.

\begin{description}
\item[check_doc] \textit{Check documentation, as \texttt{R CMD check} does.}
\end{description}

Description

This function attempts to run the documentation related checks in the same way that \texttt{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

\begin{verbatim}
check_doc(pkg = ".")
\end{verbatim}

Arguments

\begin{itemize}
\item \textbf{pkg} \quad \text{package description, can be path or package name. See \texttt{as.package} for more information}
\end{itemize}

Value

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

Examples

\begin{verbatim}
## Not run:
check_doc("mypkg")

## End(Not run)
\end{verbatim}

\begin{description}
\item[clean_dll] \textit{Remove compiled objects from \texttt{/src/} directory}
\end{description}

Description

Invisibly returns the names of the deleted files.

Usage

\begin{verbatim}
clean_dll(pkg = ".")
\end{verbatim}

Arguments

\begin{itemize}
\item \textbf{pkg} \quad \text{package description, can be path or package name. See \texttt{as.package} for more information}
\end{itemize}
**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**  
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**  
_Clean built 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**

```r
compiler_flags(debug = FALSE)
```

**Arguments**

- **debug**
  
  If TRUE adds `-g -00` 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**

```r
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.
create

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

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

setup(path = ".", description = getOption("devtools.desce"), check = FALSE,
rstudio = TRUE)

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
rstudio               Create an Rstudio project file? (with use_rstudio)

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
create_description

**Examples**

```r
## 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**

```r
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.
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 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")
exmple("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**

```r
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**

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

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

## End(Not run)
**dev_mode**

**Description**

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

**Usage**

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

**Arguments**

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

- **path**
  - directory to library.

**Examples**

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

## End(Not run)
```

---

**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**
  - package 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").
eval_clean

See Also

roxygenize, browseVignettes("roxygen2")

---

**eval_clean**  

*Evaluate code in a clean R session.*

## Description

Evaluate code in a clean R session.

## Usage

```r
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.

## 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**

- `pull` The pull request to install

**See Also**

- `install_github`

---

**has_devel**

*Check if you have a development environment installed.*

**Description**

Thanks to the suggestion of Simon Urbanek.

**Usage**

```r
has_devel()
```

**Value**

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

**Examples**

```r
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::'?'.

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

## 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")
infrastructure

## Not run:

```r
# 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)

---

### Description

Add useful infrastructure to a package.

### Usage

```r
use_testthat(pkg = ".")
use_rstudio(pkg = ".")
use_vignette(name, pkg = ".")
use_rcpp(pkg = ".")
use_travis(pkg = ".")
use_appveyor(pkg = ".")
use_package_doc(pkg = ".")
use_revdep(pkg = ".")
use_cran_comments(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.

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_rstudio
Does not modify .Rbuildignore as RStudio will do that when opened for the first time.

use_knitr
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 .Rbuildignore so it isn’t included in the built package.

use_appveyor
Add basic AppVeyor template to a package. Also adds appveyor.yml to .Rbuildignore 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.

See Also
Other infrastructure: add_build_ignore, use_build_ignore; use_data_raw; use_data; use_git_hook; use_package; use_readme_rmd
**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**

```r
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**

```r
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, build_vignettes = FALSE,
keep_source = getOption("keep.source.pkgs"), threads = getOption("Ncpus",
1))
```
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`.
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.

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_bitbucket`; `install_github`; `install_gitorious`; `install_git`; `install_svn`; `install_url`; `install_version`
install_bitbucket

Install a package directly from bitbucket

Description

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

Usage

install_bitbucket(repo, username, ref = "master", subdir = NULL,
                   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 is 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.

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.

See Also

Bitbucket API docs: https://confluence.atlassian.com/display/BITBUCKET/Use+the+Bitbucket+REST+APIs

Other package installation: install_github; install_gitorious; install_git; install_svn; install_url; install_version; install

Examples

## Not run:
install_bitbucket("sulab/mygene.r@default")
install_bitbucket("dannavaro/1sr-package")

## End(Not run)
install_deps  

Install package dependencies

**Description**

Install package dependencies

**Usage**

```r
install_deps(pkg = ".", dependencies = NA, threads = getOption("Ncpus", 1))
```

**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.

**Examples**

```r
## 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.

**Usage**

```r
install_git(url, subdir = NULL, branch = NULL, args = character(0), ...)
```

**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.
- **args**: A character vector providing extra arguments to pass on to `install`
install_github

See Also

Other package installation: install_bitbucket; install_github; install_gitorious; install_svn; install_url; install_version; install

Examples

## Not run
install_github("git://github.com/hadley/stringr.git")
install_github("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

install_github(repo, username = NULL, ref = "master", subdir = NULL, auth_token = github_pat(), host = "api.github.com", ...)

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 is 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/applications 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".

... Other arguments passed on to install.

See Also

github_pull

Other package installation: install_bitbucket; install_gitorious; install_git; install_svn; install_url; install_version; install
install_gitorious

Attempts to install a package directly from gitorious.

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

Usage
install_gitorious(repo, ref = "master", subdir = 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 is specified, the values in repo take precedence.

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.

... Other arguments passed on to install.

See Also
Other package installation: install_bitbucket; install_github; install_git; install_svn; install_url; install_version; install
install_local

Examples

```
## Not run:
install_gitorious("r-mpc-package/r-mpc-package")

## End(Not run)
```

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, ...)
```

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`.

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), ...)
```
install_url

Arguments

url Location of package. The url should point to a public or private repository.
subdir A sub-directory withing a svn 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 install
... Other arguments passed on to install

Details

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

See Also

Other package installation: install_bitbucket; install_github; install_gitorious; install_git; install_url; install_version; install

Examples

## 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(), ...)

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.
**install_version**

**See Also**

Other package installation: `install_bitbucket`; `install_github`; `install_gitorious`; `install_git`; `install_svn`; `install_version`; `install`

**Examples**

```r
## 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**

```r
install_version(package, version = NULL, repos =getOption("repos"),
    type = getOption("pkgType"), ...)
```

**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 "http://cran.us-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. Possible values are (currently) "source", "mac.binary", "mac.binary.mavericks" and "win.binary": the binary types can be listed and downloaded but not installed on other platforms.

  The default is the appropriate binary type on Windows and on the CRAN binary OS X distributions, otherwise "source". For the platforms where binary packages are the default, an alternative is "both" which means 'try binary if available and current, otherwise try source'. (This will only choose the binary package if its version number is no older than the source version. In interactive use it will ask before attempting to install source packages.)

- `...` Other arguments passed on to `install`. 
Author(s)

Jeremy Stephens

See Also

Other package installation: install_bitbucket; install_github; install_gitorious; install_git; install_svn; install_url; install

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 = ".", ...)

Arguments

pkg package description, can be path or package name. See as.package for more information
...
additional arguments passed to lint_package

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)
**Arguments**

- **pkg**
  package description, can be path or package name. See `as.package` for more information. If the DESCRIPTION file does not exist, it is created using `create_description`.

- **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.

**Details**

Currently `load_all`:

- Loads all data files in `data/`. See `load_data` for more details.
- Sources all R files in the R directory, storing results in environment that behaves like a regular package namespace. See below and `load_code` for more details.
- Compiles any C, C++, or Fortran code in the `src/` directory and connects the generated DLL into R. See `compile_dll` for more details.
- Runs `.onAttach()`, `.onLoad()` and `.onUnload()` functions at the correct times.

**Namespaces**

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

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

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

**Shim files**

`load_all` also inserts shim functions into the imports environment of the 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

```r
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

```r
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

missing_s3

Find missing s3 exports.

Description

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

Usage

missing_s3(pkg = ".")

Arguments

pkg
path

Get/set the PATH variable.

Description

Get/set the PATH variable.

Usage

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

path <- get_path()
length(path)
old <- add_path(".")
length(get_path())
set_path(old)
length(get_path())
**Description**

Run automated and manual tests, then ftp to CRAN.

**Usage**

```r
release(pkg = ".", check = TRUE)
```

**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.

**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 [http://cran.r-project.org/web/packages/policies.html](http://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**

```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.

**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 = ".")
```
revdep_check_save_logs

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)
```

revdep_check_save_logs

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 save logs with revdep_check_save_logs.

Usage

```r
revdep_check_save_logs(res, log_dir = "revdep")

revdep_check_save_summary(res, log_dir = "revdep")

revdep_check_summary(res)
```
revdep_check(pkg = ".", recursive = FALSE, ignore = NULL,
dependencies = c("Depends", "Imports", "Suggests", "LinkingTo"),
libpath = getOption("devtools.revdep.libpath"), srcpath = libpath,
bioconductor = FALSE, type = getOption("pkgType"),
threads = getOption("Ncpus", 1), check_dir = tempfile("check_cran"))

Arguments

**res**       Result of `revdep_check`

**log_dir**   Directory in which to save logs

**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.

**libpath**   Path to library to store dependencies packages - if you you’re doing this a lot it’s a good idea to pick a directory and stick with it so you don’t have to download all the packages every time.

**srcpath**   Path to directory to store source versions of dependent packages - again, this saves a lot of time because you don’t need to redownload the packages every time you run the package.

**bioconductor** If TRUE also look for dependencies amongst bioconductor packages.

**type**      binary Package type to test (source, mac.binary etc). Defaults to the same type as `install.packages()`.

**threads**   Number of concurrent threads to use for checking. It defaults to the option "Ncpus" or 1 if unset.

**check_dir** Directory to store results.

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/00check-summary.txt`.

Check process

1. Install `pkg` (in special library, see below).
2. Find all CRAN packges that dependent 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 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 `option(devtools.libpath)`.

**See Also**

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

**Examples**

```r
## Not run:
# Run R CMD check on all downstream dependencies of ggplot2
res <- revdep_check("ggplot2")
revdep_check_summary(res)
revdep_check_save_logs(res)

## 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(include_base = FALSE)

**Arguments**

include_base

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

---

**show_news**

*Show package news*

**Description**

Show package news

**Usage**

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

**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, forkable and usable as a git repository.”  

https://gist.github.com/

Usage

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

Arguments

id  
either full url (character), gist ID (numeric or character of numeric). If a gist ID is specified and the entry has multiple files, only the first R file in the gist is sourced.

...  
other options passed to source

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

# 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")

## Not run:
# Wrong hash will result in error
source_gist(6872663, sha1 = "54f1db27e61")

## End(Not run)
source_url

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

Description

Internally, source_url calls `getURL` in RCurl package and then read the contents by `textConnection`, which is then `source`ed. See ?getURL for the available protocol.

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.

Details

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.

Examples

```r
## Not run:

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

# With a hash, to make sure the remote file hasn't changed
source_url("https://gist.github.com/hadley/6872663/raw/hi.r",
  sha1 = "54f1db27e60b7e0486d785684989b49e8fe9f9")

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

## End(Not run)
```
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)
unload

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".

Details

If no testing infrastructure is present, you’ll be asked if you want devtools to create it for you (in interactive sessions only). See add_test_infrastructure for more details.

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/hadley/devtools/issues.

Usage

unload(pkg = ".")

Arguments

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

Examples

## Not run:
# Unload package that is in current directory
unload(".")

# Unload package that is in ./ggplot2/
unload("ggplot2/")

# Can use inst() to find the path of an installed package
# This will load and unload the installed ggpplot2 package
library(ggplot2)
unload(inst("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**

```
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: add_buildIgnore, use_buildIgnore; add_rstudio_project, add_test_infrastructure, add_travis, add_travis, add_travis, infrastructure, use_appveyor, use_cran_comments, use_package_doc, use_rcpp, use_revdep, use_rstudio, use_testthat, use_travis, use_vignette; use_data_raw; use_git_hook; 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: `add_build_ignore, use_build_ignore; add_rstudio_project, add_test_infrastructure, add_travis, add_travis, add_travis, infrastructure, use_appveyor, use_cran_comments, use_package_doc, use_rcpp, use_revdep, use_rstudio, use_testthat, use_travis, use_vignette; use_data; use_git_hook; 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**

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

**Arguments**

- **package**  
  Name of package to depend on.

- **type**  
  Type of dependency: must be one of "Imports", "Suggests", "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: `add_build_ignore, use_build_ignore; add_rstudio_project, add_test_infrastructure, add_travis, add_travis, add_travis, infrastructure, use_appveyor, use_cran_comments, use_package_doc, use_rcpp, use_revdep, use_rstudio, use_testthat, use_travis, use_vignette; use_data_raw; use_data; use_git_hook; use_readme_rmd`
Examples

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

## 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**

```r
with_debug(code, CFLAGS = NULL, CXXFLAGS = NULL, FFLAGS = NULL, FCFLAGS = NULL, debug = TRUE, action = "replace")
```
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
d action (for with_envvar only): should new values "replace", "suffix", "prefix"
e existing environmental variables with the same name.

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)

with_something Execute code in temporarily altered environment.

Description

• in_dir: working directory
• with_collate: collation order
• with_envvar: environmental variables
• with_libpaths: library paths, replacing current libpaths
• with_lib: library paths, prepending to current libpaths
• with_locale: any locale setting
• with_options: options
• with_path: PATH environment variable
• with_par: graphics parameters
Usage

with_envvar(new, code, action = "replace")

with_env(new, code)

with_locale(new, code)

with_collate(new, code)

in_dir(new, code)

with_libpaths(new, code)

with_lib(new, code)

with_options(new, code)

with_par(new, code)

with_path(new, code, add = TRUE)

Arguments

new values for setting
code code to execute in that environment
action (for with_envvar only): should new values "replace", "suffix", "prefix" existing environmental variables with the same name.
add Combine with existing values? Currently for with_path only. If FALSE all existing paths are overwriten, which don’t you usually want.

Deprecation

with_env will be deprecated in devtools 1.2 and removed in devtools 1.3

Examples

getwd()
in_dir(tempdir(), getwd())
getwd()

Sys.getenv("HADLEY")
with_envvar(c("HADLEY" = 2), Sys.getenv("HADLEY"))
Sys.getenv("HADLEY")

with_envvar(c("A" = 1),

  with_envvar(c("A" = 2), action = "suffix", Sys.getenv("A"))
)
INDEX

load_code, 31, 32
load_data, 31, 32
load_dll, 33

missing_s3, 33

on_path, 34
options, 13

package.skeleton, 11
path, 34

r_env_vars, 7
release, 8, 35
reload, 22, 36
revdep, 36
revdep_check, 37, 38
revdep_check(revdep_check_save_logs), 37
revdep_check_save_logs, 37
revdep_check_save_summary
  (revdep_check_save_logs), 37
revdep_check_summary
  (revdep_check_save_logs), 37
revdep_maintainers, 39
revdep_maintainers(revdep), 36
roxygenize, 15, 16
run_examples, 13, 39

save, 45
session_info, 40
sessionInfo, 40
set_path(path), 34
setup(create), 11
shim_help(help), 18
shim_question(help), 18
shim_system.file(system.file), 43
show_news, 40
source, 41, 42
source_gist, 41
source_url, 41, 42
system.file, 43, 43

test, 20, 43
test_check, 43
test_dir, 43
textConnection, 42

unload, 31, 36, 44
url, 29

use_appveyor, 45, 46
use_appveyor(infrastructure), 19
use_build_ignore, 20, 45, 46
use_cran_comments, 45, 46
use_cran_comments(infrastructure), 19
use_data, 20, 45, 46
use_data_raw, 20, 45, 46, 46
use_git_hook, 20, 45, 46
use_package, 20, 45, 46, 46
use_package_doc, 45, 46
use_package_doc(infrastructure), 19
use_rcpp, 45, 46
use_rcpp(infrastructure), 19
use_readme_rmd, 20, 45, 46
use_revdep, 45, 46
use_revdep(infrastructure), 19
use_rstudio, 11, 45, 46
use_rstudio(infrastructure), 19
use_testthat, 45, 46
use_testthat(infrastructure), 19
use_travis, 45, 46
use_travis(infrastructure), 19
use_vignette, 45, 46
use_vignette(infrastructure), 19

wd, 47
with_collate(with_something), 48
with_debug, 10, 22, 47
with_env(with_something), 48
with_envvar(with_something), 48
with_lib(with_something), 48
with_libpaths, 22
with_libpaths(with_something), 48
with_locale(with_something), 48
with_options(with_something), 48
with_par(with_something), 48
with_path, 34, 49
with_path(with_something), 48
with_something, 48