Package ‘fakemake’

November 2, 2018

Title Mock the Unix Make Utility

Version 1.4.0

Description Use R as a minimal build system. This might come in handy if you are developing R packages and can not use a proper build system. Stay away if you can (use a proper build system).

Depends R (>= 3.3.0)

License BSD_2_clause + file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/fvafrCU/fakemake

Suggests knitr, rmarkdown, testthat, RUnit, devtools, usethis, rprojroot, roxygen2, hunspell, cleanr, lintr, covr, cyclocomp, spelling

Imports MakefileR, callr, withr, utils, igraph, graphics, pkgbuild

VignetteBuilder knitr

RoxygenNote 6.1.0

NeedsCompilation no

Author Andreas Dominik Cullmann [aut, cre]

Maintainer Andreas Dominik Cullmann <fvafrcu@arcor.de>

Repository CRAN

Date/Publication 2018-11-02 20:20:04 UTC

R topics documented:

`fakemake-package` *Mock the Unix Make Utility*

Description

Use R as a minimal build system. This might come in handy if you are developing R packages and can not use a proper build system. Stay away if you can (use a proper build system).

Details

You will find the details in
`vignette("An_Introduction_to_fakemake", package = "fakemake")`.

`check_archive` *Check a Package Archive*

Description

This is a wrapper to `callr::rcmd_safe("check")`, similar to, but leaner than `rcmdcheck::rcmdcheck`. While the latter parses the output of `rcmd_safe` and uses **clisymbols** in the callback, we here just return bare output and use `writeLines` as callback. This should result in a screen display that should be identical to the output of `R CMD check`.

Usage

```
check_archive(path, cmdargs = NULL)
```

Arguments

`path`	Path to the package archive.
`cmdargs`	Command line arguments (see `callr::rcmd`)

Value

A list with the standard output, standard error and exit status of the check. (see `callr::rcmd`).

Examples

```
## Not run:
package_path <- file.path(tempdir(), "fakepack")
usethis::create_package(path = package_path)
file.copy(system.file("templates", "throw.R", package = "fakemake"),
          file.path(package_path, "R"))
roxygen2::roxygenize(package_path)
print(tarball <- get_pkg_archive_path(package_path))
devtools::build(pkg = package_path, path = package_path)
print(check_archive(tarball))

## End(Not run)
```

`check_archive_as_cran` *A Convenience Wrapper to* `check_archive`

Description

A Convenience Wrapper to `check_archive`

Usage

```
check_archive_as_cran(path)
```

Arguments

`path`	Path to the package archive.

`get_pkg_archive_path` *Get a Package's Archive Path From the Package's DESCRIPTION*

Description

Get a Package's Archive Path From the Package's DESCRIPTION

Usage

```
get_pkg_archive_path(path = ".", absolute = TRUE)
```

Arguments

`path`	Path to the package.
`absolute`	Return the absolute path?

Value

Path to the package's archive file.

Note

The archive file does not have to exist. Use `file.exists(get_pkg_archive_path())` to test existence.

Examples

```
package_path <- file.path(tempdir(), "anRpackage")
usethis::create_package(path = package_path)
print(tarball <- get_pkg_archive_path(package_path))
file.exists(tarball)
```

`make` *Mock the Unix Make Utility*

Description

Mock the Unix Make Utility

Usage

```
make(name, make_list, force = FALSE, recursive = force,
  verbose = TRUE)
```

Arguments

`name`	The name or alias of a make target.
`make_list`	The `makelist` (a listed version of a `Makefile`).
`force`	Force the target to be build?
`recursive`	Force the target to be build recursively (see *Note*)?
`verbose`	Be verbose?

Value

A character vector containing the targets made during the current run.

Note

Forcing a target mocks adding .PHONY to a GNU `Makefile` if you set recursive to FALSE. If recursive is TRUE, then the whole make chain will be forced.

Examples

```
str(make_list <- provide_make_list("minimal"))
# build all
withr::with_dir(tempdir(), print(make("all.Rout", make_list)))
# nothing to be done
withr::with_dir(tempdir(), print(make("all.Rout", make_list)))
# forcing all.Rout
withr::with_dir(tempdir(), print(make("all.Rout", make_list, force = TRUE,
                                      recursive = FALSE)))
# forcing all.Rout recursively
withr::with_dir(tempdir(), print(make("all.Rout", make_list, force = TRUE)))
```

`provide_make_list` *Load an Example* `Makelist` *Provided by* **fakemake**.

Description

Load an Example `Makelist` Provided by **fakemake**.

Usage

```
provide_make_list(type = c("minimal", "package", "standard"),
  prune = TRUE, clean_sink = FALSE)
```

Arguments

`type`	The type of `makelist`. Use "standard" for a standard package `makelist`.
`prune`	Prune the `makelist` of `NULL` items?
`clean_sink`	Remove sinks identical to corresponding targets from the list? Since `makelists` are parsed, missing sinks are set to the corresponding targets, but this makes them harder to read.

Value

A `makelist`.

Examples

```
str(provide_make_list("minimal"))
```

read_makefile — *Read a* Makefile *Into a* Makelist

Description

Read a Makefile Into a Makelist

Usage

```
read_makefile(path, clean_sink = FALSE)
```

Arguments

path	The path to the file.
clean_sink	Remove sinks identical to corresponding targets from the list? Since makelists are parsed, missing sinks are set to the corresponding targets, but this makes them harder to read.

Value

The makelist.

Note

This function will not read arbitrary Makefiles, just those created via write_makefile! If you modify such a Makefile make sure you only add simple rules like the ones you see in that file.

Examples

```
make_file <- file.path(tempdir(), "Makefile")
write_makefile(provide_make_list(), path = make_file)
str(make_list <- read_makefile(path = make_file))
```

sink_all — *Divert Message And Output Stream to File*

Description

All output and messages up to the first error, for example thrown by stop.

Usage

```
sink_all(path, code)
```

Arguments

`path`	The path of the file to divert to.
`code`	The code to be executed.

Value

`Invisibly` `NULL`.

Examples

```
sink_path  <- file.path(tempdir(), "sink_all.txt")
sink_all(sink_path, {
         print("some output")
         warning("a warning")
         message("a message")
         print("some more output")
})
cat(readLines(sink_path), sep = "\n")
```

`touch` *Mock the Unix* `touch` *utility*

Description

Creating a file or ensuring a file's modification time changes.

Usage

```
touch(path)
```

Arguments

`path`	Path to the file to be touched

Value

The return value of `file.copy`.

Examples

```
file <- tempfile()
touch(file)
t1 <- file.mtime(file)
touch(file)
t2 <- file.mtime(file)
t1 < t2
```

visualize *Parse a* `Makelist`*, Convert it Into an* `Igraph` *and Plot it*

Description

Parse a `Makelist`, Convert it Into an `Igraph` and Plot it

Usage

```
visualize(make_list, root = NULL)
```

Arguments

`make_list`	The `makelist`.
`root`	The root of a tree.

Value

Invisibly an **igraph** representation of the `makelist`.

Examples

```
str(ml <- provide_make_list("package"))
visualize(ml)
visualize(ml, root = "log/check.Rout")
```

write_makefile *Write a* `Makelist` *to File*

Description

The `makelist` is parsed before writing, so all R code which is not in a "code" item will be evaluated. So if any other item's string contains code allowing for a dynamic rule, for example with some "dependencies" reading `"list.files(\"R\", full.names = TRUE)"`, the `Makefile` will have the evaluated code, a list static list of files in the above case.

Usage

```
write_makefile(make_list, path, Rbin = "Rscript-devel")
```

Arguments

`make_list`	The list to write to file.
`path`	The path to the file.
`Rbin`	The R binary to use in the `Makefile`.

Value

See MakefileR::write_makefile.

Examples

```
make_file <- file.path(tempdir(), "my_Makefile")
write_makefile(provide_make_list(), path = make_file)
cat(readLines(make_file), sep = "\n")
```

Index