

# Package ‘fakemake’

March 31, 2022

**Title** Mock the Unix Make Utility

**Version** 1.10.4

**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).

**License** BSD\_2\_clause + file LICENSE

**URL** <https://gitlab.com/fvafrcu/fakemake>

**Depends** R (>= 3.3.0)

**Imports** graphics, fritools, igraph, MakefileR, utils

**Suggests** cleanr, covr, cyclocomp, devtools, hunspell, knitr, lintr, packager (>= 1.5.0), pkgbuild, pkgload, rasciidoc, rmarkdown, roxygen2, rprojroot, RUnit, spelling, testthat, withr, usethis

**VignetteBuilder** knitr, rasciidoc

**Encoding** UTF-8

**RoxygenNote** 7.1.1

**NeedsCompilation** no

**Author** Andreas Dominik Cullmann [aut, cre]

**Maintainer** Andreas Dominik Cullmann <fvafrcu@mailbox.org>

**Repository** CRAN

**Date/Publication** 2022-03-31 12:50:02 UTC

## R topics documented:

fakemake-package . . . . .	2
add_target . . . . .	2
get_target . . . . .	3
make . . . . .	4
provide_make_list . . . . .	5
remove_target . . . . .	6
visualize . . . . .	7

<b>Index</b>	<b>8</b>
--------------	----------

---

fakemake-package	<i>Mock the Unix Make Utility</i>
------------------	-----------------------------------

---

### 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").

---

add_target	<i>Add a Target to a Makelist</i>
------------	-----------------------------------

---

### Description

Add a target to an existing makelist.

### Usage

```
add_target(
  makelist,
  target,
  code,
  prerequisites = NULL,
  prerequisite_to = NULL,
  sink = NULL,
  alias = sub("\\.(Rout|log)$", "", basename(target))
)
```

### Arguments

makelist	A list for <a href="#">make</a> .
target	The target to remove from makelist.
code	The code for the new target.
prerequisites	The prerequisites for the new target.
prerequisite_to	The targets the new target is a prerequisite to. Set to <a href="#">TRUE</a> to add it as a prerequisite to all existing targets.
sink	The sink for the new target.
alias	The alias for the new target.

**Value**

A list for [make](#).

**See Also**

Other functions to manipulate makelists: [get\\_target\(\)](#), [remove\\_target\(\)](#)

---

get\_target

*Get a Makelist's Target*

---

**Description**

Get a single target from a makelist by alias.

**Usage**

```
get_target(makelist, alias)
```

**Arguments**

makelist      A list for [make](#).  
alias          The alias of the target in question.

**Value**

A list (the target requested).

**See Also**

Other functions to manipulate makelists: [add\\_target\(\)](#), [remove\\_target\(\)](#)

**Examples**

```
m1 <- provide_make_list()
visualize(m1, root = "all.Rout")
i <- which(sapply(m1, "[[", "target") == "b1.Rout")
m1[[i]][["alias"]] <- "b1"
t <- get_target(m1, "b1")
m1 <- remove_target(m1, t[["target"]])
visualize(m1)
m1 <- add_target(m1, target = t[["target"]], code = t[["code"]],
                 sink = t[["sink"]],
                 prerequisite_to = "a1.Rout", alias = NULL)
all.equal(m1, provide_make_list())
```

---

`make`*Mock the Unix Make Utility*

---

## Description

Mock the Unix Make Utility

## Usage

```
make(  
  name,  
  make_list,  
  force = FALSE,  
  recursive = force,  
  verbose = FALSE,  
  verbosity = 2,  
  dry_run = FALSE,  
  unconditionally = FALSE,  
  stop_on_warning = FALSE  
)
```

## Arguments

<code>name</code>	The name or alias of a make target.
<code>make_list</code>	The makelist (a listed version of a Makefile).
<code>force</code>	Force the target to be build? See <i>Details</i> .
<code>recursive</code>	Force the target to be build recursively? See <i>Details</i> .
<code>verbose</code>	Be verbose?
<code>verbosity</code>	Give the level of verbosity.
<code>dry_run</code>	Run dry? Mock GNU make's -n option.
<code>unconditionally</code>	Force the target's code to be evaluated unconditionally to any prerequisites? See <i>Details</i> .
<code>stop_on_warning</code>	<a href="#">Throw an error and abort</a> if a recipe throws a <a href="#">warning</a> ?

## Details

`force, recursive`

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.

`unconditionally`

Setting `unconditionally` to `TRUE` allows you to fool `make` similarly to using GNU make's `-touch` option.

**Value**

**Invisibly** a character vector containing the targets made during the current run.

**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)))

# show files
dir(tempdir(), pattern = ".*\\.Rout")

# dry run
file.remove(dir(tempdir(), pattern = ".*\\.Rout", full.names = TRUE))
withr::with_dir(tempdir(), print(make("all.Rout", make_list,
                                     dry_run = TRUE)))
dir(tempdir(), pattern = ".*\\.Rout")

# make unconditionally
dir(tempdir(), pattern = ".*\\.Rout")
withr::with_dir(tempdir(), print(make("all.Rout", make_list,
                                     unconditionally = TRUE)))
dir(tempdir(), pattern = ".*\\.Rout")
```

---

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", "testing", "vignette"),
  prune = TRUE,
  clean_sink = FALSE
)
```

**Arguments**

type	The type of makelist. 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"))
visualize(provide_make_list("minimal"))
```

---

remove_target	<i>Remove a Target From a Makelist</i>
---------------	--

---

**Description**

Remove a target and all its appearances as other targets' dependencies from a makelist.

**Usage**

```
remove_target(makelist, target)
```

**Arguments**

makelist	A list for <a href="#">make</a> .
target	The target to remove from makelist.

**Value**

A list for [make](#).

**See Also**

Other functions to manipulate makelists: [add\\_target\(\)](#), [get\\_target\(\)](#)

---

`visualize`*Visualize a Makelist*

---

**Description**

Parse a makelist, convert it into an igraph and plot it.

**Usage**

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

**Arguments**

<code>make_list</code>	The makelist.
<code>root</code>	The root of a tree.

**Value**

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

**Examples**

```
str(ml <- provide_make_list())  
visualize(ml)  
visualize(ml, root = "all.Rout")
```

# Index

## \* functions to manipulate makelists

- add\_target, 2
- get\_target, 3
- remove\_target, 6

## \* package

- fakemake-package, 2

add\_target, 2, 3, 6

fakemake-package, 2

get\_target, 3, 3, 6

Invisibly, 5

make, 2-4, 4, 6

provide\_make\_list, 5

remove\_target, 3, 6

Throw an error and abort, 4

TRUE, 2, 4

visualize, 7

warning, 4