Package: staged.dependencies (via r-universe)

October 7, 2024

Type Package

Title Install R packages from Particular Git Branches

Version 0.3.1.9001

Imports checkmate, desc, devtools, digest, dplyr, fs, git2r, glue, httr, jsonlite, methods, rcmdcheck, remotes, rlang, stats, tidyr, utils, withr, yaml,

Description When developing multiple dependent packages, it is often useful to introduce development stages (devel, pre-release, release) that synchronize these packages. This package provides an implementation of development stages via branch naming rules. It defines RStudio addins that allow to install the matching upstream and downstream dependencies.

License file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Suggests knitr, miniUI, mockery, renv, rmarkdown, rstudioapi, shiny, testthat (>= 2.1.0), visNetwork

VignetteBuilder knitr

Collate 'argument_convention.R' 'caching.R' 'dependencies.R' 'dependencies_app.R' 'dependencies_helper.R' 'git_tools.R' 'graph_methods.R' 'host.R' 'ref_strategy.R' 'renv.R' 'rstudio_jobs.R' 'rstudio_addins.R' 'utils.R' 'zzz.R'

Repository https://pharmaverse.r-universe.dev

RemoteUrl https://github.com/openpharma/staged.dependencies

RemoteRef HEAD

RemoteSha fb124997306b35d44a0225bb4b400bf7258c4c75

Contents

`build_check_install` *Build, check and install internal dependencies*

Description

Build, check and install internal dependencies

Usage

```
build_check_install(
  dep_structure,
  install_direction = "all",
  steps = c("build", "check", "install"),
 rcmd_args = list(check = c("--no-multiarch", "--with-keep.source", "--install-tests")),
  artifact_dir = tempfile(),
  install_external_deps = TRUE,
  upgrade = "never",
  package_list = NULL,
  dry = FALSE,
  verbose = 1,
  ...
)
```

Arguments

dep_structure	(dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies
install_direction	"upstream", "downstream" or "all"; which packages to install (according to dependency structure). By default this is only "upstream"
steps	(character vector) subset of "build", "check", "install"; useful to skip checking for example
rcmd_args	(list) with names build, check, install which are vectors that are passed as separate arguments to the R CMD commands
artifact_dir	(character) directory to place built R packages and logs
install_external_deps	logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) .
upgrade	argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment.
package_list	(character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only).
dry	(logical) dry run that outputs what would happen without actually doing it.
verbose	(numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)
...	Arguments passed on to install_deps install_project (logical) whether to also install the current package (i.e. the package named in dependency_structure$current_pkg), ignored unless install_direction = "upstream" (because downstream deps automatically install all their upstream deps)

Value

list with entries

- artifact_dir: artifact_dir directory with log files
- pkg_actions: data.frame of performed actions

Examples

```
## Not run:
x <- dependency_table(project = ".", verbose = 1)
build_check_install(x, steps = c("build", "check"), verbose = 1)
build_check_install(x, artifact_dir = "../output")

## End(Not run)
```

check_downstream *Check & install downstream dependencies*

Description

Installs downstream R packages as specified in a dependency_structure object and then runs rcmdcheck (R CMD check) on the downstream dependencies.

Usage

```
check_downstream(
  dep_structure,
  distance = NULL,
  check_args = c("--no-multiarch", "--with-keep.source", "--install-tests"),
  only_tests = FALSE,
  install_external_deps = TRUE,
  upgrade = "never",
  package_list = NULL,
  dry = FALSE,
  verbose = 1,
  ...
)
```

Arguments

dep_structure	(dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies
distance	(numeric) additional filter to only install downstream packages at most this distance from the dependency_structure$current_pkg (advanced use only)
check_args	(list) arguments passed to rcmdcheck
only_tests	(logical) whether to only run tests (rather than checks)
install_external_deps	logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) .
upgrade	argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment.
package_list	(character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only).
dry	(logical) dry run that outputs what would happen without actually doing it.
verbose	(numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)

... Arguments passed on to install_deps

install_project (logical) whether to also install the current package (i.e. the package named in dependency_structure$current_pkg), ignored unless install_direction = "upstream" (because downstream deps automatically install all their upstream deps)

install_direction "upstream", "downstream" or "all"; which packages to install (according to dependency structure). By default this is only "upstream"

Value

data.frame of performed actions

Examples

```
## Not run:
x <- dependency_table(project = ".", verbose = 1)

check_downstream(x, verbose = 1)
check_downstream(x, verbose = 1, only_test = TRUE, check_args = c("--no-manual"))

## End(Not run)
```

---

check_downstream_job *Check & install downstream job*

Description

Check & install downstream job

Usage

```
check_downstream_job(
  project = ".",
  verbose = 1,
  create_args = list(renv_profile = Sys.getenv("RENV_PROFILE")),
  ...
)
```

Arguments

project (character) If project_type is local then directory of project (for which to calculate the dependency structure); must be a git repository. If project_type is repo@host then should be character of the form openpharma/stageddeps.food@https://github.co If host is not included in the string then the default https://github.com is assumed.

verbose (numeric)
verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)

create_args named list - additional arguments passed to dependency_table function

... Arguments passed on to check_downstream

distance (numeric) additional filter to only install downstream packages at most this distance from the dependency_structure$current_pkg (advanced use only)

check_args (list) arguments passed to rcmdcheck

only_tests (logical) whether to only run tests (rather than checks)

dep_structure (dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies

install_external_deps logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) .

upgrade argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment.

package_list (character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only).

dry (logical) dry run that outputs what would happen without actually doing it.

See Also

check_downstream

Examples

```
## Not run:
check_downstream_job(check_args = Sys.getenv("RCMDCHECK_ARGS"))
check_downstream_job(
  check_args = Sys.getenv("RCMDCHECK_ARGS"),
  list(create_arg = list(ref = "6_makegraph@main"))
)
check_downstream_job(only_tests = TRUE)

## End(Not run)
```

---

check_yamls_consistent

Checks that the staged dependency yamls are consistent with the dependencies listed in the DESCRIPTION files

---

Description

Checks that the staged dependency yamls are consistent with the dependencies listed in the DESCRIPTION files

Usage

```
check_yamls_consistent(dep_structure, skip_if_missing_yaml = FALSE)
```

Arguments

dep_structure	dependency_structure object
skip_if_missing_yaml	logical should checks be skipped on packages without yaml files. Default FALSE

Details

This function explicitly checks that for all packages in the dependency_structure object: all upstream and downstream packages specified in each yaml file are found in the appropriate package DESCRIPTION file

Value

NULL if successful. An error is thrown if inconsistencies found

Examples

```
## Not run:
x <- dependency_table(project = ".")
check_yamls_consistent(x)

## End(Not run)
```

---

clear_cache *Clear the repository cache*

Description

Use this function to clear the package cache of some or all repositories (depending on pattern) if the git operations fail.

Usage

```
clear_cache(pattern = "*")
```

Arguments

pattern	files to remove, see unlink (wildcards * and ? allowed)

Examples

```
## Not run:
clear_cache()
clear_cache("*elecinfra*")

## End(Not run)
```

`dependency_table` *Create dependency structure of your package collection*

Description

Create dependency structure of your package collection

Usage

```
dependency_table(
  project = ".",
  project_type = c("local", "repo@host")[1],
  ref = NULL,
 local_repos = if ((project_type) == "local") get_local_pkgs_from_config() else NULL,
  direction = "all",
  fallback_branch = "main",
  renv_profile = NULL,
  verbose = 1
)
```

Arguments

`project`	(`character`) If `project_type` is `local` then directory of project (for which to calculate the dependency structure); must be a git repository. If `project_type` is `repo@host` then should be character of the form `openpharma/stageddeps.food@https://github.co` If host is not included in the string then the default `https://github.com` is assumed.
`project_type`	(`character`) See `project` argument.
`ref`	(`character`) git branch (or tag) inferred from the branch of the project if not provided; warning if not consistent with current branch of project. If `project_type` is not `local` then this argument must be provided.
`local_repos`	(`data.frame`) repositories that should be taken from local file system rather than cloned; columns are `repo, host, directory`.
`direction`	(`character`) direction in which to discover packages either "upstream","downstream" or "all".
`fallback_branch`	(`character`) the default branch to try to use if no other matches found. It defaults to `"main"`.

`renv_profile` (`character`) the name of the renv profile of the `renv.lock` files to be included from the repos. The standard `renv.lock` file uses the default `NULL` argument here.

`verbose` (`numeric`)
verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)

Value

`dependency_structure` An S3 object with the following items:

project `project` argument used to create the object (absolute path if `project_type` is `local`

project_type `project_type` used to create object

current_pkg The R package name of code in the `project` directory

table `data.frame` contain one row per r package discovered, with the following rows `package_name`, `type` (`current`, `upstream`, `downstream` or `other`), `distance` (minimum number of steps from `current_pkg`), `ref`, `repo`, `host`, `sha` `cache_dir`, `accessible`, `installable` and `install_index` (the order to install the packages). Note some items are are suppressed when printing the object

deps `list` with three elements, `upstream_deps`is the graph where edges point from a package to its upstream dependencies. They are ordered in installation order. The `downstream_deps` list is the graph with the edge direction flipped, and is ordered in reverse installation order. `external` contains the external R packages found in the description files of the internal packages. It is a dataframe of the form returned by `desc::desc_get_deps`

direction `direction` argument used to create object

renv_files `named list` containing the json of the renv.lock files for the chosen profile for each repo. An entry to the list is `NULL` if a repos does not have the required lock file

Examples

```
## Not run:
dependency_table(verbose = 1)
dependency_table(
  project = "openpharma/stageddeps.food@https://github.com",
  project_type = "repo@host",
  ref = "main"
)
x <- dependency_table(
  project = "path/to/project",
  direction = c("upstream")
)
print(x)
plot(x)

## End(Not run)
```

`determine_ref` *Determine the branch/tag to install based on feature (staging rules)*

Description

Return the git ref (tag or branch) of the repo to install given the available branches and tags.

Usage

```
determine_ref(ref, available_refs, fallback_branch = "main", branch_sep = "@")
```

Arguments

`ref`	ref we want to build
`available_refs`	data.frame with columns `ref` the names of the available refs and `type` (`branch` or `tag`)
`fallback_branch`	the default branch to try to use if no other matches found
`branch_sep`	separator between branches in `feature`, `/` does not work well with `git` because it clashes with the filesystem paths

Details

A ref is either a tag or branches separated by slashes of the form `name1@name2@...@nameN`. Where separator is specified by `branch_sep` argument

This function checks for an exact match for the tag if this is not found then among the available branches, it searches in the order `name1@name2@...@nameN`, `name2@name3@...@nameN`, `name3@name4@...@nameN`, ..., `nameN`

Value

branch/tag to choose to match feature, error if no suitable branch was provided with the type attribute "tag" or "branch"

Examples

```
determine_ref(
  "feature1",
  data.frame(ref = c("main", "feature1"), type = "branch")
) == structure("feature1", type = "branch")

determine_ref(
  "feature1@devel",
  data.frame(ref = c("main", "devel", "feature1"), type = "branch")
) == structure("devel", type = "branch")

determine_ref(
```

```
  ref = "fix1@feature1@devel",
  available_refs = data.frame(
    ref = c(
      "main", "devel", "feature1", "feature1@devel",
      "fix1@feature1@devel", "fix1"
    ),
    type = "branch"
  )
) == structure("fix1@feature1@devel", type = "branch")

determine_ref(
  "fix1@feature1@devel",
  data.frame(
    ref = c("main", "devel", "feature1", "feature1@devel", "fix1"),
    type = "branch"
  )
) == structure("feature1@devel", type = "branch")

determine_ref(
  "fix1@feature1@devel",
  data.frame(ref = c("main", "devel", "feature1", "fix1"), type = "branch")
) == structure("devel", type = "branch")

determine_ref("feature1@release", data.frame(ref = c("main", "devel"), type = "branch"))

# error because neither `feature1@release` nor `release` branch exists
# determine_ref("feature1@release", data.frame(ref = c("master", "devel"), type = "branch"))

# tag examples
determine_ref(
  "v0.1",
 data.frame(ref = c("main", "devel", "feature1", "v0.1"), type = c(rep("branch", 3), "tag"))
) == structure("v0.1", type = "tag")

determine_ref(
  "v0.2",
 data.frame(ref = c("main", "devel", "feature1", "v0.1"), type = c(rep("branch", 3), "tag"))
) == structure("main", type = "branch")
```

`get_all_external_dependencies` *List the external R packages required to be installed*

Description

List the external R packages required to be installed

Usage

```
get_all_external_dependencies(
  dep_structure,
  available_packages = as.data.frame(utils::available.packages()),
  install_direction = "upstream",
  package_list = NULL,
  from_internal_dependencies = c("Depends", "Imports", "LinkingTo", "Suggests"),
  from_external_dependencies = c("Depends", "Imports", "LinkingTo")
)
```

Arguments

dep_structure	(dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies
available_packages	(data.frame) A dataframe of the format given by as.data.frame(utils::available.packages). It is unlikely this default needs to be changed; however you need to ensure the options("repos") contains the urls of all expected repos (e.g. Bioconductor).
install_direction	"upstream", "downstream" or "all"; which packages to install (according to dependency structure). By default this is only "upstream"
package_list	(character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only).
from_internal_dependencies	Vector chosen from c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances") which fields of the DESCRIPTION file of the internal packages should be included. Default: c("Depends", "Imports", "LinkingTo", "Suggests")
from_external_dependencies	Vector chosen from c("Depends", "Imports", "LinkingTo", "Suggests", "Enhances") which fields of the DESCRIPTION file of the internal packages should be included. Default: c("Depends", "Imports", "LinkingTo")

Value

A vector of 'external' R packages required to install the selected 'internal' packages, ordered by install order (unless from_external_dependencies does not include "Depends", "Imports" and "LinkingTo"). The core R packages (e.g. methods, utils) are not included. The output can be used with remotes::system_requirements to extract the system requirements needed for your packages, see example below.

Examples

```
## Not run:
x <- dependency_table("openpharma/stageddeps.electricity",
  project_type = "repo@host", feature = "main"
)
```

```
# get external package dependencies
ex_deps <- get_all_external_dependencies(x)
print(ex_deps)

# get system dependencies (in this case there are none)
unique(unlist(lapply(ex_deps,
  function(pkg, ...) {
    remotes::system_requirements(package = pkg, ...)
  },
  os = "ubuntu",
  os_release = "20.04"
)))

## End(Not run)
```

`get_local_pkgs_from_config` *Loads the config file and extracts* `local_packages`

Description

Checks that all directories exist and are absolute paths.

Usage

```
get_local_pkgs_from_config()
```

Value

local_packages

Examples

```
get_local_pkgs_from_config()
```

`install_deps` *Install dependencies of project*

Description

Given a `dependency_structure` object, install the R packages

Usage

```
install_deps(
  dep_structure,
  install_project = TRUE,
  install_direction = "upstream",
  install_external_deps = TRUE,
  upgrade = "never",
  package_list = NULL,
  dry = FALSE,
  verbose = 1,
  ...
)
```

Arguments

dep_structure	(dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies
install_project	(logical) whether to also install the current package (i.e. the package named in dependency_structure$current_pkg), ignored unless install_direction = "upstream" (because downstream deps automatically install all their upstream deps)
install_direction	"upstream", "downstream" or "all"; which packages to install (according to dependency structure). By default this is only "upstream"
install_external_deps	logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) .
upgrade	argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment.
package_list	(character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only).
dry	(logical) dry run that outputs what would happen without actually doing it.
verbose	(numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)
...	Additional args passed to remotes::install_deps(). Ignored if inside an renv environment.

Value

data.frame of performed actions

See Also

determine_branch

Examples

```
## Not run:
x <- dependency_table(project = "./path/to/project")

install_deps(x)

# install all dependencies
install_deps(x, install_direction = "all")

## End(Not run)
```

`install_deps_app` *Gadget or Shiny app to select the dependencies to install*

Description

The dependencies are obtained by traversing the upstream and downstream repositories in the package's staged dependencies yaml files starting from `project`.

Usage

```
install_deps_app(
  default_repo = NULL,
  default_host = "https://github.com",
  default_ref = "main",
  fallback_branch = "main",
  run_gadget = TRUE,
  run_as_job = TRUE,
  verbose = 1,
  install_external_deps = TRUE,
  renv_profile = NULL,
  upgrade = "never",
  ...
)
```

Arguments

`default_repo`	(`character`) the repository name for the dependency graph to be created for, for example, `"openpharma/stageddeps.water"`. If `NULL` this must be entered by app user and can always be changed by the user.
`default_host`	(`character`) the host for the repository for the dependency graph to be created for by default `"https://github.com"`. If `NULL` this must be entered by app user and can always be changed by the user.

default_ref (character) default ref (branch/tag), see also the parameter ref of \link{dependency_table}. If NULL this must be entered by app user and can always be changed by the user.

fallback_branch (character) the default branch to try to use if no other matches found

run_gadget (logical) whether to run the app as a gadget

run_as_job (logical) whether to run the installation as an RStudio job.

verbose (numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)

install_external_deps logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) .

renv_profile (character) the name of the renv profile of the renv.lock files to be included from the repos. The standard renv.lock file uses the default NULL argument here.

upgrade argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment.

... Additional args passed to remotes::install_deps(). Ignored if inside an renv environment.

Value

shiny.app or value returned by app (executed as a gadget)

Examples

```
## Not run:
install_deps_app("openpharma/stageddeps.food")

## End(Not run)
```

---

install_deps_job *Install dependencies job*

Description

Install dependencies job

Usage

```
install_deps_job(
  project = ".",
  project_type = "local",
  verbose = 1,
  create_args = list(renv_profile = Sys.getenv("RENV_PROFILE")),
  ...
)
```

Arguments

project	(character) If project_type is local then directory of project (for which to calculate the dependency structure); must be a git repository. If project_type is repo@host then should be character of the form openpharma/stageddeps.food@https://github.co If host is not included in the string then the default https://github.com is assumed.
project_type	(character) See project argument.
verbose	(numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)
create_args	named list - additional arguments passed to dependency_table function
...	Arguments passed on to install_deps dep_structure (dependency_structure) output of function dependency_table; uses dep_structure$table to infer the packages to apply action to and infer installation order; uses dep_structure$deps to infer upstream dependencies install_project (logical) whether to also install the current package (i.e. the package named in dependency_structure$current_pkg), ignored unless install_direction = "upstream" (because downstream deps automatically install all their upstream deps) install_direction "upstream", "downstream" or "all"; which packages to install (according to dependency structure). By default this is only "upstream" install_external_deps logical to describe whether to install external dependencies of package using remotes::install_deps() (or renv::install() if inside an renv environment) . upgrade argument passed to remotes::install_deps(), defaults to 'never'. Ignored if inside an renv environment. package_list (character) If not NULL, an additional filter, only packages on this list will be considered and their dependencies installed if needed (advanced usage only). dry (logical) dry run that outputs what would happen without actually doing it.

See Also

install_deps

Examples

```
## Not run:
install_deps_job()
install_deps_job(create_args = list(ref = "6_makegraph@main"))

# install all dependencies
install_deps_job(create_args = list(direction = "all"))
install_deps_job(dry_install = TRUE)

## End(Not run)
```

install_repo_add_sha *Install a git repository*

Description

It adds the git SHA to the DESCRIPTION file, so that the package does not need to be installed again when the same commit is already installed.

Usage

```
install_repo_add_sha(repo_dir, ...)
```

Arguments

repo_dir	directory of repo
...	Additional args passed to remotes::install_deps. Note upgrade is set to "never" and shouldn't be passed into this function.

topological_sort *Topological graph sort*

Description

Graph is a list which for each node contains a vector of child nodes in the returned list, parents appear before their children.

Usage

```
topological_sort(graph)
```

Arguments

graph	(named list) list with node vector elements mapping from child to its parents (upstream dependencies)

Details

Implementation of Kahn algorithm with a modification to maintain the order of input elements.

Value

vector listing parents before children

Examples

```
staged.dependencies:::topological_sort(list(A = c(), B = c("A"), C = c("B"), D = c("A")))
staged.dependencies:::topological_sort(list(D = c("A"), A = c(), B = c("A"), C = c("B")))
staged.dependencies:::topological_sort(list(D = c("A"), B = c("A"), C = c("B"), A = c()))
## Not run:
# cycle
topological_sort(list(A = c("B"), B = c("C", "A"), C = c()))

## End(Not run)
```

`update_with_direct_deps` *Update existing stage_dependencies yaml file*

Description

Using the existing stage_dependencies yaml file 'graph' to define internal dependencies, update the project yaml file to include to include all direct (i.e. distance 1) upstream and downstream repos

Usage

```
update_with_direct_deps(dep_structure)
```

Arguments

`dep_structure`	`dep_structure` object, output of `dependency_table` function with `project_type = "local"`

`verbose_sd_option` *Set staged.dependencies verbosity*

Description

Functions to set and remove the option parameter `verbose_level_staged.deps`. It can assume integer values between `c(0, 1, 2)`. This will set this variable as an option with `options()` and `getOption()`.

Usage

```
verbose_sd_set(verbose = 1)

verbose_sd_get()

verbose_sd_rm()
```

Arguments

verbose	(numeric) verbosity level, incremental; (0: None, 1: packages that get installed + high-level git operations, 2: includes git checkout infos)

Examples

```
verbose_sd_set(2)
verbose_sd_get() # 2, the inserted value
verbose_sd_rm()
verbose_sd_get() # 1, the default
```

Index