Package 'packer'

October 14, 2022

Title An Opinionated Framework for Using 'JavaScript'

Date 2022-05-28

Version 0.1.3

Description
Enforces good practice and provides convenience functions to make work with 'JavaScript' not just easier but also scalable. It is a robust wrapper to 'NPM', 'yarn', and 'webpack' that enables to compartmentalize 'JavaScript' code, leverage 'NPM' and 'yarn' packages, include 'TypeScript', 'React', or 'Vue' in web applications, and much more.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.0

Imports fs, cli, usethis, jsonlite, roxygen2, rprojroot, rstudioapi, assertthat, htmlwidgets

URL https://github.com/JohnCoene/packer, https://packer.john-coene.com

BugReports https://github.com/JohnCoene/packer/issues

Suggests testthat, covr, golem

NeedsCompilation no

Author John Coene [aut, cre] (<https://orcid.org/0000-0002-6637-4107>)

Maintainer John Coene <jcoenep@gmail.com>

Repository CRAN

Date/Publication 2022-05-28 09:30:02 UTC

R topics documented:

`add_plugin_clean` *Clean Plugin*

Description

Add the clean-webpack-plugin to clean the bundled files.

Usage

```
add_plugin_clean(dry = FALSE, verbose = FALSE, clean = TRUE, protect = TRUE)
```

Arguments

`dry`	Whether to simulate the removal of files.
`verbose`	Write Logs to the console.
`clean`	Whether to automatically remove all unused webpack assets on rebuild.
`protect`	Whether to not allow removal of current webpack assets.

`add_plugin_eslint` *ESLint Plugin*

Description

Add the eslint-webpack-plugin run ESLint on files.

Usage

```
add_plugin_eslint()
```

add_plugin_html *HTML Plugin*

Description

Add the html-webpack-plugin to the configuration to generate HTML with webpack, used in packer to generate the UI of a golem app with webpack.

Usage

```
add_plugin_html(use_pug = FALSE, output_path = "../index.html")
```

Arguments

use_pug	Set to TRUE to use the pug engine.
output_path	Path to the generated html file, defaults to ../index.html as is ideal for golem. Note that this path is relative to your output directory specified in your webpack.common.js file.

add_plugin_prettier *Prettier Plugin*

Description

Add the prettier-webpack-plugin to prettify the pre-bundled files.

Usage

```
add_plugin_prettier()
```

add_plugin_workbox *Progressive Web Applications*

Description

Add the workbox-webpack-plugin to the config files.

Usage

```
add_plugin_workbox()
```

apply_framework7 *Apply Framework7*

Description

Apply Framework7 to a project, adds the relevant (babel) loader, installs dependencies, and creates, or updates, or replaces the `srcjs/index.js` file.

Usage

```
apply_framework7()
```

Details

After running this function and bundling the JavaScript remember to place `div(id = "app"), tags$script(src = "www/in` at the bottom of your shiny UI.

apply_react *Apply React*

Description

Apply React to a project, adds the relevant (babel) loader, installs dependencies, and creates, updates, or replaces the `srcjs/index.js` file.

Usage

```
apply_react(use_cdn = TRUE)
```

Arguments

`use_cdn`	Whether to use the CDN for `react` and `react-dom` (recommended). This means later importing the dependencies in the shiny UI using `reactCDN()`, this function will be created in a `R/react_cdn.R`. The correct instructions to do so are printed to the console by the function.

Details

After running this function and bundling the JavaScript remember to place the code printed by the function in shiny UI. By default `apply_react()` does not bundle `react` and `react-dom` and thus requires using `reactCDN()` to import the dependencies in the shiny application: this function is created in a `R/react_cdn.R`.

apply_vue *Apply Vue*

Description

Apply Vue to a project, adds the relevant (babel) loader, installs dependencies, and creates, or updates, or replaces the `srcjs/index.js` file.

Usage

```
apply_vue(use_cdn = TRUE)
```

Arguments

`use_cdn`	Whether to use the CDN for `vue` (recommended). This means later importing the dependencies in the shiny UI using `vueCDN()`, this function will be created in a `R/vue_cdn.R`. The correct instructions are printed to the console by the application.

Details

After running this function and bundling the JavaScript remember to place `div(id = "app"), tags$script(src = "www/in` at the bottom of your shiny UI.

bundle *bundle & Watch*

Description

Bundle and watch the JavaScript.

Usage

```
bundle(mode = c("production", "development", "none"))

bundle_prod()

bundle_dev()

watch()
```

Arguments

`mode`	The configuration mode tells webpack to use its built-in optimisations accordingly.

Functions

- `bundle()` - bundle the project.
- `bundle_prod()` - bundle the project optimising production, equivalent to `bundle("production")` and `npm run production`.
- `bundle_dev()` - bundle the project including debugging developer tools, equivalent to `bundle("development")` and `npm run development`.
- `watch()` - watches for changes in the `srcjs` and rebuilds if necessary, equivalent to `npm run watch`.

---

checks *Checks*

Description

Run checks on a package using packer.

Usage

```
checks()
```

Checks

- Output files are minified
- put_precommit_hook is in place
- put_rprofile_adapt is in place

---

dev_roclet *Roclet Dev*

Description

Roclet to run bundle_dev when documenting.

Usage

```
dev_roclet()
```

ease_lit *Lit*

Description

Use Lit in your project.

Usage

```
ease_lit(ts = FALSE)
```

Arguments

ts	Whether to use TypeScript (recommended).

engine *Set Engine*

Description

Defines the engine to use with packer. One can pick between npm and yarn.

Usage

```
engine_set(engine = c("npm", "yarn"))

engine_get()

engine_adapt()

engine_which()
```

Arguments

engine	The engine to use, npm or yarn.

Details

Generally one would want to define the engine prior to scaffolding. For convenience you can instead set the environment variable `PACKER_ENGINE` to your engine of choice. Packer reads this variable, all subsequent use of packer will use the defined engine. You can use the function `usethis::edit_r_environ` to do so.

Functions

- `engine_set`: Define the engine to use for the project.
- `engine_get`: Retrieve the default engine.
- `engine_which`: Retrieve which engine the project is set to use-.
- `engine_adapt`: Change the engine to match that of the poject.

`engine_check` *Engine Check*

Description

Check if the engine is correctly set up and prints helpful messages if not.

Usage

```
engine_check()
```

`engine_console` *Npm Output*

Description

Prints the output of the last npm command run, useful for debugging.

Usage

```
engine_console()
```

`include_action_check` *Github Actions*

Description

Adds a Github Action to the package that will ensure JavaScript files have been bundled for production.

Usage

```
include_action_check()
```

jsdoc *Add Plugin jsdoc*

Description

Add the jsdoc plugin to generate documentation from JavaScript code with tags similar to roxygen2.

Usage

```
add_plugin_jsdoc(edit = interactive())

add_jsdoc_tutorial(name, edit = interactive())
```

Arguments

edit	Whether to open relevent file.
name	Name of tutorial

make_library *Make Library*

Description

Adds library settings to webpack config. This allow exporting JavaScript objects.

Usage

```
make_library(name = "[name]", type = "umd")
```

Arguments

name	Name of the library, default recommended, see details.
type	Type of the library.

Details

The functions will be exported at the specified `name`, e.g.: if the name is `myLib` then functions can be called with `myLib.function();`. The default (`[name]`) means the name of the exported library will be the same as the name of the scaffold. This is advised because otherwise, if one has multiple scaffold, an absolute will overwrite itself and only the last scaffold added will be a valid library.

mockup *Mock up*

Description

Functions to mock up packages for tests

Usage

```
tmp_package()

tmp_golem()

tmp_project()

tmp_ambiorix()

tmp_delete(tmp)
```

Arguments

tmp	A temp mock up project.

npm_console *Npm Output*

Description

Prints the output of the last npm command run, useful for debugging.

Usage

```
npm_console()
```

npm_fix *Audit Fix*

Description

Scan your project for vulnerabilities and automatically install any compatible updates to vulnerable dependencies.

Usage

```
npm_fix()
```

Details

Runs `npm audit fix`

npm_install *Install and Uninstall Npm Packages*

Description

Install and uninstall npm packages.

Usage

```
npm_install(..., scope = c("dev", "prod", "global"))

npm_uninstall(..., scope = c("dev", "prod", "global"))
```

Arguments

`...`	Packages to install or uninstall. If no packages are specified then this function install packages in `package.json` (useful e.g.: after clone).
`scope`	Scope of installation or uninstallation, see scopes.

Scopes

- `prod` - Installs/Uninstalls packages for project with `--save`
- `dev` - Installs/Uninstalls dev packages for project with `--save-dev`
- `global` - Installs/Uninstalls packages globally with `-g`

npm_outdated *Npm Outdated*

Description

Find outdated dependencies

Usage

```
npm_outdated()
```

npm_run *Npm Command*

Description

Convenience function to run `npm` commands.

Usage

```
npm_run(...)
```

Arguments

`...`	Passed to `system2()`.

npm_update *Npm Update*

Description

Update npm dependencies.

Usage

```
npm_update()
```

npx *NPX*

Description

Run an npx command.

Usage

```
npx(...)
```

Arguments

...	Arguments to pass to npx

prod_roclet *Roclet Prod*

Description

Roclet to run bundle_prod when documenting.

Usage

```
prod_roclet()
```

put_precommit_hook *Put Pre-Commit Hook*

Description

Add a pre-commit hook that runs at every commit to ensure that JavaScript files are minified.

Usage

```
put_precommit_hook()
```

Note

Will only work if using git.

Examples

```
## Not run: put_precommit_hook()
```

put_recommended *Recommended Checks*

Description

Recommended checks for packer projects, runs put_rprofile_adapt and put_precommit_hook.

Usage

```
put_recommended()
```

put_rprofile_adapt *Rprofile*

Description

Add `engine_adapt()` to `.Rprofile`.

Usage

```
put_rprofile_adapt()
```

Details

This is recommended so anyone contributing to the project is guaranteed to be on the correct engine.

put_test *Put a Test*

Description

Puts a testthat test to ensure the files are optimised for prod.

Usage

```
put_test()
```

Note

This function adds packer to `Suggests`.

scaffold_ambiorix *Ambiorix*

Description

Creates the basic structure for an ambiorix application.

Usage

```
scaffold_ambiorix(vue = FALSE, use_cdn = TRUE, edit = interactive())
```

Arguments

vue	Whether to include Vue, internally runs `apply_vue()` and adapts the `srcjs/index.js` template for Vue.
use_cdn	Whether to use the CDN for react or vue dependencies, this is passed to `apply_react()` or `apply_vue()` if `react` or `vue` arguments are set to `TRUE` and ignored otherwise.
edit	Automatically open pertinent files.

Details

Only one of `react` or `vue` can be set to `TRUE`.

Value

`TRUE` (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_ambiorix()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_ambiorix()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

scaffold_bare *Bare*

Description

Creates a bare scaffold for not specific use case, as opposed to other scaffolds. This scaffold does not generate R code.

Usage

```
scaffold_bare(edit = interactive())
```

Arguments

edit	Automatically open pertinent files.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_package()

# move to package
setwd(tmp)

# scaffold bare
scaffold_bare()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

scaffold_extension *Shiny Extension*

Description

Creates the basic structure for a shiny extension.

Usage

```
scaffold_extension(name, edit = interactive())
```

Arguments

name	Name of extension used to define file names and functions.
edit	Automatically open pertinent files.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_package()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_extension()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

scaffold_golem *Golem*

Description

Creates the basic structure for golem app with JavaScript.

Usage

```
scaffold_golem(
  react = FALSE,
  vue = FALSE,
  framework7 = FALSE,
  use_cdn = TRUE,
  edit = interactive()
)
```

Arguments

react	Whether to include React, internally runs apply_react() and adapts the srcjs/index.js template for React.
vue	Whether to include Vue, internally runs apply_vue() and adapts the srcjs/index.js template for Vue.
framework7	Whether to include Framework7, internally runs apply_framework7() and adapts the srcjs/index.js template for Framework7.
use_cdn	Whether to use the CDN for react, vue or Framework7 dependencies, this is passed to apply_react(), apply_vue() or apply_framework7() if react, vue or framework7 arguments are set to TRUE and ignored otherwise.
edit	Automatically open pertinent files.

Details

Only one of react, vue or framework7 can be set to TRUE. use_cdn is not supported for Framework7.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up golem project
tmp <- tmp_golem()
```

```
# move to package
setwd(tmp)

# scaffold golem
scaffold_golem()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

`scaffold_input` *Scaffold a Custom Input*

Description

Sets basic structure for a shiny input.

Usage

```
scaffold_input(name, edit = interactive())
```

Arguments

`name`	Name of input, will define internal name binding and CSS class.
`edit`	Automatically open pertinent files.

Value

`TRUE` (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_package()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_input()

# clean up
```

```
setwd(wd)
tmp_delete(tmp)
}
```

`scaffold_leprechaun` *Leprechaun*

Description

Creates the basic structure for leprechaun app with JavaScript.

Usage

```
scaffold_leprechaun(
  react = FALSE,
  vue = FALSE,
  use_cdn = TRUE,
  edit = interactive()
)
```

Arguments

`react`	Whether to include React, internally runs `apply_react()` and adapts the `srcjs/index.js` template for React.
`vue`	Whether to include Vue, internally runs `apply_vue()` and adapts the `srcjs/index.js` template for Vue.
`use_cdn`	Whether to use the CDN for react or vue dependencies, this is passed to `apply_react()` or `apply_vue()` if `react` or `vue` arguments are set to `TRUE` and ignored otherwise.
`edit`	Automatically open pertinent files.

Details

Only one of `react` or `vue` can be set to `TRUE`.

Value

`TRUE` (invisibly) if successfully run.

scaffold_output *Scaffold Shiny Output*

Description

Sets basic structure for a shiny input.

Usage

```
scaffold_output(name, edit = interactive())
```

Arguments

name	Name of output, will define internal name binding and CSS class.
edit	Automatically open pertinent files.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_package()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_output()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

scaffold_rmd *Golem*

Description

Creates the basic structure for golem app with JavaScript.

Usage

```
scaffold_rmd(react = FALSE, vue = FALSE, edit = interactive())
```

Arguments

react	Whether to include React, internally runs apply_react() and adapts the srcjs/index.js template for React.
vue	Whether to include Vue, internally runs apply_vue() and adapts the srcjs/index.js template for Vue.
edit	Automatically open pertinent files.

Details

Only one of react or vue can be set to TRUE.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_project()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_rmd()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

scaffold_widget *Scaffold Widget*

Description

Creates basic structure for a widget.

Usage

```
scaffold_widget(name, edit = interactive())
```

Arguments

name	Name of widget, also passed to `htmlwidgets::scaffoldWidget()`.
edit	Automatically open pertinent files.

Details

Internally runs `htmlwidgets::scaffoldWidget()` do not run it prior to this function.

Value

TRUE (invisibly) if successfully run.

Examples

```
if(interactive()){
# current directory
wd <- getwd()

# create a mock up ambiorix project
tmp <- tmp_package()

# move to package
setwd(tmp)

# scaffold ambiorix
scaffold_widget()

# clean up
setwd(wd)
tmp_delete(tmp)
}
```

set_npm *Set npm path*

Description

By default packer looks for the npm installation using the `which` (or `where`) command. This function lets you override that behaviour and force a specific npm installation.

Usage

```
set_npm(path = NULL)
```

Arguments

`path`	Path to npm installation to use.

set_yarn *Set yarn path*

Description

By default packer looks for the yarn installation using the `which` (or `where`) command. This function lets you override that behaviour and force a specific yarn installation.

Usage

```
set_yarn(path = NULL)
```

Arguments

`path`	Path to yarn installation to use.

tests *Add Tests*

Description

Adds tests to a project, currently supports mocha and peeky, see details for more.

Usage

```
include_tests_mocha(esm = TRUE)

include_tests_peeky()

add_test_file(name)

run_tests(open = FALSE)
```

Arguments

esm	Whether to install esm and require it for tests (recommended).
name	Name of the test file to add, without extension.
open	Only valid for "peeky," this will open a development UI if TRUE.

Details

include_tests_mocha uses mocha and mocha-webpack and creates a directory called testjs where tests should be placed. The function run_tests() will then uses mocha on all the files in the testjs directory. All tests should end with .test.js. include_tests_peeky uses peeky it's very similar to mocha but also comes with a development UI that can be accessed when running tests by setting open to TRUE.

Requiring esm (esm = TRUE) is recommended as it will allow using the latest ESM, e.g.: import in tests.

types *Install Types*

Description

Install TypeScript types from npm.

Usage

```
ts_get_types(..., versions = NULL)

ts_get_type(type, version = NULL)
```

Arguments

`...`	Types to install.
`versions, version`	Corresponding versions of types passed to `...`, if `NULL` the latest version is installed.
`type`	Name of types `@types/*` to install.

Functions

- `ts_get_types`: Flexible to retrieve multiple types.
- `ts_get_type`: Convenience to easily retrieve a single type.

Examples

```
## Not run: ts_get_type("jquery")
## Not run: ts_get_types("@types/jquery")
```

`use_loader_babel` *Use babel Loader*

Description

Adds the loader for babel compiler to the loader configuration file.

Usage

```
use_loader_babel(test = "\\.(js|jsx)$", use_eslint = FALSE)
```

Arguments

`test`	Test regular expression test which files should be transformed by the loader.
`use_eslint`	Whether to also add the ESlint loader.

Details

The `use_elsint` argument is useful here as loaders have to be defined in the correct order or files might be checked after being processed by babel.

Excludes `node_modules` by default.

use_loader_coffee *Use Coffee Loader*

Description

Adds the coffee-loader to use cofeescript.

Usage

```
use_loader_coffee(test = "\\.coffee$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

Details

Excludes node_modules by default.

use_loader_file *Use File Loader*

Description

Adds the file-loader to resolve files: png, jpg, jpeg, and gif.

Usage

```
use_loader_file(test = "\\.(png|jpe?g|gif)$/i")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

use_loader_framework7 *Use Framework7 Loader*

Description

Adds the Framework7 loader.

Usage

```
use_loader_framework7(test = "\\.(f7).(html|js|jsx)$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

Details

Excludes node_modules by default. If used outside scaffold_golem context, installs the babel-loader in the dev scope.

use_loader_mocha *Use Mocha Loader*

Description

Adds the mocha-loader for tests.

Usage

```
use_loader_mocha(test = "\\.test\\.js$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

Details

Excludes node_modules by default.

use_loader_pug *Use Pug Loader*

Description

Adds the loader for the pug templating engine.

Usage

```
use_loader_pug(test = "\\.pug$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

use_loader_rule *Add a Loader rule*

Description

Adds a loader rule that is not yet implemented in packer.

Usage

```
use_loader_rule(
  packages,
  test,
  ...,
  use = as.list(packages),
  .name_use = "use"
)
```

Arguments

packages	NPM packages (loaders) to install.
test	Test regular expression test which files should be transformed by the loader.
...	Any other options to pass to the rule.
use	Name of the loaders to use for `test`.
.name_use	Depending on the webpack config one might want to change the `use` to `loader` or `loaders`.

Details

Reads the `srcjs/config/loaders.json` and appends the rule.

use_loader_style *Use Styles Loader*

Description

Installs loaders and adds relevant configuration rules to `srcjs/config/loaders.json`, the function `use_loader_style` is *recommended*.

Usage

```
use_loader_css(test = "\\.css$", import = TRUE, modules = TRUE)

use_loader_sass(test = "\\.s[ac]ss$/i")

use_loader_style(test = "\\.css$", import = TRUE, modules = TRUE)
```

Arguments

`test`	Test regular expression test which files should be transformed by the loader.
`import`	Whether to enable `import` statements for `.css` files. If `FALSE` use `require`.
`modules`	Enables CSS modules and their config, a complex but powerful feature detailed here

Details

This will let you import styles much like any other modules, e.g.: `import './styles.css'`.

Packages

- `use_loader_css()` - installs and imports `css-loader` packages as dev.
- `use_loader_style()` - installs and imports `style-loader` and `css-loader` packages as dev. This loader enabled CSS modules.
- `use_loader_sass()` - installs and imports `style-loader`, `css-loader`, and `sass-loader` as dev.

use_loader_svelte *Use Svelte Loader*

Description

Add the loader svelte

Usage

```
use_loader_svelte(test = "\\.(html|svelte)$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

---

use_loader_ts *Use Typescript Loader*

Description

Adds the loader for the pug templating engine.

Usage

```
use_loader_ts(test = "\\.tsx?$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

---

use_loader_vue *Use Vue Loader*

Description

Adds the Vue loader to the loader configuration file.

Usage

```
use_loader_vue(test = "\\.vue$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

Details

Every time a new version of Vue is released, a corresponding version of `vue-template-compiler` is released together. The compiler's version must be in sync with the base Vue package so that `vue-loader` produces code that is compatible with the runtime. This means every time you upgrade Vue in your project, you should upgrade `vue-template-compiler` to match it as well.

use_tailwind *Use Tailwind*

Description

Creates PostCSS, and tailwindcss config files as well as adds the appropriate loaders and installs dependencies.

Usage

```
use_tailwind(test = "\\.css$")
```

Arguments

test	Test regular expression test which files should be transformed by the loader.

yarn_clean *Yarn cache clean*

Description

Clean the cache

Usage

```
yarn_clean()
```

Value

The semver as a string.

Examples

```
## Not run: yarn_clean()
```

yarn_console *Yarn Output*

Description

Prints the output of the last command run, useful for debugging.

Usage

```
yarn_console()
```

yarn_global *Yarn Global*

Description

Installs or manage yarn *globally*.

Usage

```
engine_yarn_install()

engine_yarn_set(version = "latest")
```

Arguments

version	Version to set yarn

Functions

- engine_yarn_install: Installs yarn globally.
- engine_yarn_set: Set yarn function.

Examples

```
## Not run: engine_yarn_install()
```

yarn_install *Install and Uninstall yarn Packages*

Description

Install and uninstall yarn packages.

Usage

```
yarn_add(..., scope = c("dev", "prod"))

yarn_install()

yarn_remove(..., scope = c("dev", "prod"))
```

Arguments

...	Packages to install or uninstall.
scope	Scope of installation or uninstallation, see scopes.

Scopes

- `prod` - Add/remove packages for project with no flag
- `dev` - Installs/Uninstalls dev packages for project with `--dev`

Examples

```
## Not run: yarn_add("browserify")
```

---

yarn_outdated *Yarn Outdated*

Description

Find outdated dependencies

Usage

```
yarn_outdated()
```

Examples

```
## Not run: yarn_outdated()
```

---

yarn_run *Yarn Command*

Description

Convenience function to run `yarn` commands.

Usage

```
yarn_run(...)
```

Arguments

`...`	Passed to `system2()`.

yarn_upgrade *Yarn Upgrade*

Description

Upgrade yarn dependencies.

Usage

```
yarn_upgrade()
```

Examples

```
## Not run: yarn_upgrade()
```

yarn_version *Npm version*

Description

Get the version of npm.

Get the version of yarn.

Usage

```
yarn_version()

yarn_version()
```

Value

The semver as a string.

The semver as a string.

Examples

```
## Not run: yarn_version()
```

Index