rp

for version 0.1, December 20, 2021

C. Dardis <christopherdardis@gmail.com>

Table of Contents

1 Preamble

`lexical binding` is `t`.
Copyright: 2021-2022 - Chris Dardis
Author: C. Dardis <christopherdardis@gmail.com>
URL: `http://github.com/dardisco/rp`
Version: 0.1
Keywords: R, Sweave, knitr, latex, Rnw, bash
Commentary: Building and checking an 'R' package can be tedious and typically requires a variety of tools, both within 'R' and on the command line. The 'rp' package aims to do the needful with a single function!

The package assumes that you are using the following workflow: - Writing an 'R' package that follows best practice guidelines for submission to CRAN. - Using 'roxygen2' for package documentation. - Using 'bash' for shell commands. This *may* work with other shell types.

On calling the function 'rp-rp', Emacs will open a new buffer to display output from R and shell processes. The functions on 'rp-function-sequence' are then run. This performs the following steps: - Ensure 'R' is the latest stable version (update if necessary) - Ensure all 'R' packages are the latest version. This can take a long time if you have many packages! - Use 'roxygen2' to update the package documentation. Note that you will still need to update the following files, as necessary: * DESCRIPTION * NEWS * README.md * The package documentation file e.g. 'packageName.R' which has a line in 'roxygen' format like "#' @docType package". - Build the package using 'R CMD build'. This will take place in the parent directory of the package. - Check the package using 'R CMD check –as-cran'. - If 'rp-check-rud' is 't', also check the package with 'R Under development'. This is the latest, unstable, release of 'R'. This last step is not recommended routinely but typically should be performed once it has passed all of the package checks performed by the stable version of 'R'.

Installation:

This package can be installed using

(package-install-file "/path/to/rp.el")

Or place the folliwng in your init.el file:

(add-to-list 'load-path "~/path/to/directory") (require 'rp)

Usage:

To start the process use the 'rp-rp' command, e.g. with 'M-x mp-mp RET' This command should be called within the directory tree of the 'R' package. E.g. this can be called from a buffer which is a file in the package or from a 'dired' buffer which is in the package.

Many of the functions in the package can be called independently. For example, you may wish to run 'rp-check-examples' before going through the whole sequence, as this is a common source of ERRORs and WARNINGs.

For developers:

Function-local/temporary variables are named using as name1 e.g. 'v1', 'buffer1'.

2 defgroup

Documentation for `defgroup`:
Declare SYMBOL as a customization group containing MEMBERS. SYMBOL does not need to be quoted.

Third argument DOC is the group documentation. This should be a short description of the group, beginning with a capital and ending with a period. Words other than the first should not be capitalized, if they are not usually written so.

MEMBERS should be an alist of the form ((NAME WIDGET)...) where NAME is a symbol and WIDGET is a widget for editing that symbol. Useful widgets are 'custom-variable' for editing variables, 'custom-face' for editing faces, and 'custom-group' for editing groups.

The remaining arguments should have the form

[KEYWORD VALUE]...

For a list of valid keywords, see the common keywords listed in 'defcustom'.

See Info node '(elisp) Customization' in the Emacs Lisp manual for more information.

A table of all `defgroup`s follows:

`rp`

Details of each `defgroup` follow below:

rp

`rp` [defgroup]

custom-group
((rp-function-sequence custom-variable) (rp-check-rud custom-variable) (rp-path-to-rud custom-variable) (rp-rud-sh custom-variable))

group-documentation
R package. This is a series of variables and functions to simplity the process of building and checking an R package for submission to CRAN.

custom-prefix
rp-

custom-version
0.1

3 defun

Documentation for `defun`:
Define NAME as a function. The definition is (lambda ARGLIST [DOCSTRING] BODY...). See also the function ‘interactive’. DECL is a declaration, optional, of the form (declare DECLS...) where DECLS is a list of elements of the form (PROP . VALUES). These are interpreted according to ‘defun-declarations-alist’. The return value is undefined.

(fn NAME ARGLIST &optional DOCSTRING DECL &rest BODY)

A table of all `defun`s follows:

`rp-rp`	Run the list of functions in ‘rp-function-sequence’.
`rp-update-R`	Update R and packages, if necessary.
`rp-shell-sentinel`	Run COMMAND as a shell process and pause until done.
`rp-roxy-write-doc`	Use roxygen to write documentation.
`rp-get-DESCRIPTION-parent`	Get name of parent directory of DESCRIPTION.
`rp-get-DESCRIPTION`	Get the DESCRIPTION file for a package.
`rp-get-package-name`	Get the package name from the DESCRIPTION file.
`rp-get-version`	Get the version from the DESCRIPTION file.
`rp-check-examples`	Check package examples using devtools.
`rp-build-package`	Build package using R CMD build.
`rp-check-package`	Check package.
`rp-update-rud`	Update ’R Under development’, the development version of R.
`rp-log`	Open the log file.

Details of each `defun` follow below:

rp-rp

`rp-rp` [Function]

Command ? *yes*

Arguments: *none*

Documentation: Run the list of functions in ‘rp-function-sequence’.

```
((interactive)
 (pop-to-buffer "*R-package*")
 (erase-buffer)
 (let
     ((--dolist-tail-- rp-function-sequence))
   (while --dolist-tail--
     (let
          ((x
            (car --dolist-tail--)))
       (funcall x)
       (setq --dolist-tail--
             (cdr --dolist-tail--))))))
```

rp-update-R

`rp-update-R` [Function]

Command ? *yes*

Arguments: *none*

Documentation: Update R and packages, if necessary.

```
((interactive)
 (rp-shell-sentinel "Rscript -e 'if(!(require(installr)))\n    install.packages('installr', dependencies=
 (rp-shell-sentinel "Rscript -e 'update.packages(ask=FALSE, instlib=.libPaths()[1])'"))
```

rp-shell-sentinel

`rp-shell-sentinel` [Function]

Command ? *no*

Arguments: *COMMAND*

Documentation: Run COMMAND as a shell process and pause until done. Output appears in buffer '*R-package*'.

```
((let
     ((--cl-buffer1--
       (make-symbol "--buffer1--"))
      (--cl-command1--
       (make-symbol "--command1--")))
   (progn
     (let*
          ((v --cl-buffer1--))
       (set v "*R-package*"))
     (let*
          ((v --cl-command1--))
       (set v COMMAND)))
   (progn
     (pop-to-buffer
      (symbol-value --cl-buffer1--))
(set-process-sentinel
 (start-process-shell-command
  (symbol-value --cl-buffer1--)
  (symbol-value --cl-buffer1--)
  (symbol-value --cl-command1--))
 (list 'lambda
         '(&rest --cl-rest--)
         (list 'apply
                 (list 'function
                         #'(lambda
                             (G4 G5 process1 event1)
                             (if
                                   (= 0
                                      (process-exit-status process1))
                                   (progn
                                     (message
                                      (concat "rp-shell-sentinel, with COMMAND "
                                                (symbol-value G4)
                                                " ... done"))
                                     (throw 'exit nil))
                               (progn
                                   (message
                                    (concat "Error in rp-shell-sentinel, with COMMAND "
                                              (symbol-value G4))))))
                 (list 'quote --cl-command1--)
                 (list 'quote --cl-buffer1--)
                 '--cl-rest--)))
(recursive-edit))))
```

rp-roxy-write-doc

`rp-roxy-write-doc` [Function]

Command ? *yes*

Arguments: *none*

Documentation: Use roxygen to write documentation. Uses .R files with roxygen markup to create .Rd help files.

```
((interactive)
 (let*
     ((description1
       (rp-get-DESCRIPTION))
      (package1
       (rp-get-package-name description1))
      (command1
       (concat "Rscript -e \"setwd('"
                 (concat
                    (rp-get-DESCRIPTION-parent description1)
                    "')\nif(!(require(roxygen2)))\n    install.packages('roxygen2', dependencies=TRUE)\nr
   (rp-shell-sentinel command1)))
```

rp-get-DESCRIPTION-parent

`rp-get-DESCRIPTION-parent` [Function]

Command ? *no*

Arguments: *DESCRIPTION*

Documentation: Get name of parent directory of DESCRIPTION.

```
((file-name-directory
  (directory-file-name
   (file-name-directory DESCRIPTION))))
```

rp-get-DESCRIPTION

`rp-get-DESCRIPTION` [Function]

Command ? *yes*

Arguments: *none*

Documentation: Get the DESCRIPTION file for a package. Search upwards from the ‘default-directory’ to find the first file named DESCRIPTION.

```
((interactive)
 (catch 'done
   (let
       ((d1
           (car
            (directory-files default-directory t "^DESCRIPTION$"))))
     (if d1 nil
       (setq d1
               (locate-dominating-file
                (if
                      (buffer-file-name)
                      (buffer-file-name)
                    default-directory)
                "DESCRIPTION")))
     (if d1
           (progn
             (setq d1
                     (car
                      (directory-files d1 t "^DESCRIPTION$")))))
     (if d1 nil
       (message "No DESCRIPTION file on this directory tree")
       (throw 'done nil))
     (setq d1
             (file-truename d1))
     (message
      (prin1-to-string d1))
     d1)))
```

rp-get-package-name

`rp-get-package-name` [Function]

Command ? *yes*

Arguments: *DESCRIPTION*

Documentation: Get the package name from the DESCRIPTION file.

```
((interactive)
 (find-file DESCRIPTION)
 (search-forward "package: " nil t)
 (let
     ((pn1
       (buffer-substring-no-properties
          (point)
          (line-end-position))))
   (replace-regexp-in-string " " "" pn1)
   (kill-buffer
    (buffer-name))
   pn1))
```

rp-get-version

`rp-get-version` [Function]

Command ? *yes*

Arguments: *DESCRIPTION*

Documentation: Get the version from the DESCRIPTION file.

```
((interactive)
 (find-file DESCRIPTION)
 (search-forward "version: " nil t)
 (let
     ((vn1
       (buffer-substring-no-properties
          (point)
          (line-end-position))))
   (replace-regexp-in-string " " "" vn1)
   (kill-buffer
    (buffer-name))
   vn1))
```

rp-check-examples

`rp-check-examples` [Function]

Command ? *yes*

Arguments: *none*

Documentation: Check package examples using devtools.

```
((interactive)
 (let*
     ((description1
       (rp-get-DESCRIPTION))
      (package1
       (rp-get-package-name description1))
      (command1
       (concat "Rscript -e \"setwd('"
                 (concat
                    (rp-get-DESCRIPTION-parent description1)
                    "')\nif(!(require(devtools)))\n    install.packages('devtools', dependencies=TRUE)\nde
   (rp-shell-sentinel command1)))
```

rp-build-package

rp-build-package [Function]

Command ? *yes*

Arguments: *none*

Documentation: Build package using R CMD build.

```
((interactive)
 (let*
     ((description1
       (rp-get-DESCRIPTION))
      (package1
       (rp-get-package-name description1))
      (command1
       (concat "cd "
               (rp-get-DESCRIPTION-parent description1)
               " && R CMD build " package1)))
   (rp-shell-sentinel command1)))
```

rp-check-package

rp-check-package [Function]

Command ? *yes*

Arguments: *none*

Documentation: Check package. Checks a tarball, e.g. as built by ‘rp-build-package’. Uses R CMD check –as-cran. If ‘rp-check-rud’ is ‘t’, also check with R Under development.

```
((interactive)
 (let*
     ((description1
       (rp-get-DESCRIPTION))
      (package1
       (rp-get-package-name description1))
      (version1
       (rp-get-version description1))
      (command1
       (concat "cd "
               (rp-get-DESCRIPTION-parent description1)
               " && R CMD check --as-cran " package1 "_" version1 ".tar.gz")))
   (rp-shell-sentinel command1)
   (if rp-check-rud
       (progn
         (rp-update-rud)
         (setq command1
               (concat "cd "
                       (rp-get-DESCRIPTION-parent description1)
                       " && " rp-path-to-rud "bin/R CMD check --as-cran " package1 "_" version1 ".tar
         (rp-shell-sentinel command1)))))
```

rp-update-rud

`rp-update-rud` [Function]

Command ? *yes*
Arguments: *none*
Documentation: Update 'R Under development', the development version of R.

```
((interactive)
 (let
     ((ov1
       (string-to-number
          (shell-command-to-string "svn info https://svn.r-project.org/R/trunk |\ngrep --regexp='Revision
      (lv1
       (string-to-number
          (shell-command-to-string "command -v 'rud' && rud --version |\ngrep --only-matching --regexp='r
   (if
       (< lv1 ov1) (progn (rp-shell-sentinel (concat "mkdir --parents " rp-path-to-rud)) (rp-shell-sentin
 @end smallformat
@end defun

@cindex rp-log
@unnumberedsec rp-log
@defun rp-log
Command ? @emph{yes} @*
Arguments: @emph{none} @*
Documentation: Open the log file.

@smallformat @verbatim
((interactive)
 (pop-to-buffer "*R-package*")
 (goto-char
  (point-max))
 (let
     ((log1
       (search-backward-regexp "for details.$" nil t)))
   (if log1
       (progn
           (setq log1
                 (replace-regexp-in-string "[ `'\n]" ""
                                                (buffer-substring-no-properties
                                                 (point)
                                                 (line-end-position -1))))
           (insert-file-contents log1)))))
```

4 defcustom

Documentation for `defcustom`:
Declare SYMBOL as a customizable variable. SYMBOL is the variable name; it should not be quoted. STANDARD is an expression specifying the variable's standard value. It should not be quoted. It is evaluated once by 'defcustom', and the value is assigned to SYMBOL if the variable is unbound. The expression itself is also stored, so that Customize can re-evaluate it later to get the standard value. DOC is the variable documentation.

This macro uses 'defvar' as a subroutine, which also marks the variable as "special", so that it is always dynamically bound even when 'lexical-binding' is t.

The remaining arguments to 'defcustom' should have the form

[KEYWORD VALUE]...

The following keywords are meaningful:

:type VALUE should be a widget type for editing the symbol's value. Every 'defcustom' should specify a value for this keyword. :options VALUE should be a list of valid members of the widget type. :initialize VALUE should be a function used to initialize the variable. It takes two arguments, the symbol and value given in the 'defcustom' call. The default is 'custom-initialize-reset'. :set VALUE should be a function to set the value of the symbol when using the Customize user interface. It takes two arguments, the symbol to set and the value to give it. The function should not modify its value argument destructively. The default choice of function is 'set-default'. :get VALUE should be a function to extract the value of symbol. The function takes one argument, a symbol, and should return the current value for that symbol. The default choice of function is 'default-value'. :require VALUE should be a feature symbol. If you save a value for this option, then when your init file loads the value, it does (require VALUE) first. :set-after VARIABLES Specifies that SYMBOL should be set after the list of variables VARIABLES when both have been customized. :risky Set SYMBOL's 'risky-local-variable' property to VALUE. :safe Set SYMBOL's 'safe-local-variable' property to VALUE. See Info node '(elisp) File Local Variables'.

The following common keywords are also meaningful.

:group VALUE should be a customization group. Add SYMBOL (or FACE with 'defface') to that group. :link LINK-DATA Include an external link after the documentation string for this item. This is a sentence containing an active field which references some other documentation.

There are several alternatives you can use for LINK-DATA:

(custom-manual INFO-NODE) Link to an Info node; INFO-NODE is a string which specifies the node name, as in "(emacs)Top".

(info-link INFO-NODE) Like 'custom-manual' except that the link appears in the customization buffer with the Info node name.

(url-link URL) Link to a web page; URL is a string which specifies the URL.

(emacs-commentary-link LIBRARY) Link to the commentary section of LIBRARY.

(emacs-library-link LIBRARY) Link to an Emacs Lisp LIBRARY file.

(file-link FILE) Link to FILE.

(function-link FUNCTION) Link to the documentation of FUNCTION.

(variable-link VARIABLE) Link to the documentation of VARIABLE.

(custom-group-link GROUP) Link to another customization GROUP.

You can specify the text to use in the customization buffer by adding ':tag NAME' after the first element of the LINK-DATA; for example, (info-link :tag "foo" "(emacs)Top") makes a link to the Emacs manual which appears in the buffer as 'foo'.

An item can have more than one external link; however, most items have none at all. :version VALUE should be a string specifying that the variable was first introduced, or its default value was changed, in Emacs version VERSION. :package-version VALUE should be a list with the form (PACKAGE . VERSION) specifying that the variable was first introduced, or its default value was changed, in PACKAGE version VERSION. This keyword takes priority over :version. For packages which are bundled with Emacs releases, the PACKAGE and VERSION must appear in the alist 'customize-package-emacs-version-alist'. Since PACKAGE must be unique and the user might see it in an error message, a good choice is the official name of the package, such as MH-E or Gnus. :tag LABEL Use LABEL, a string, instead of the item's name, to label the item in customization menus and buffers. :load FILE Load file FILE (a string) before displaying this customization item. Loading is done with 'load', and only if the file is not already loaded.

If SYMBOL has a local binding, then this form affects the local binding. This is normally not what you want. Thus, if you need to load a file defining variables with this form, or with 'defvar' or 'defconst', you should always load that file _outside_ any bindings for these variables. ('defvar' and 'defconst' behave similarly in this respect.)

See Info node '(elisp) Customization' in the Emacs Lisp manual for more information.

A table of all `defcustom`s follows:

`rp-function-sequence`
: This 'list' defines the functions to be called when 'rp-rp' is called.

`rp-check-rud`
: Also check package with 'R Under development'?

`rp-path-to-rud`
: The path to install 'R Under development'?

`rp-rud-sh`
: Shell script to install R Under development.

Details of each `defcustom` follow below:

rp-function-sequence

`rp-function-sequence` [User Option]

standard-value

((funcall (function (closure (t) nil (quote (rp-update-R rp-roxy-write-doc rp-check-examples rp-build-package rp-check-package))))))

custom-type

(list)

custom-requests
nil

variable-documentation
This ‘list’ defines the functions to be called when ‘rp-rp’ is called.

rp-check-rud

`rp-check-rud` [User Option]

standard-value
((funcall (function (closure (t) nil nil))))

custom-type
(radio (const `:doc` No `:value` nil) (const `:doc` Yes `:value` t))

custom-requests
nil

variable-documentation
Also check package with ’R Under development’?

rp-path-to-rud

`rp-path-to-rud` [User Option]

standard-value
((funcall (function (closure (t) nil $HOME/rud/))))

custom-type
(directory)

custom-requests
nil

variable-documentation
The path to install ’R Under development’?

rp-rud-sh

`rp-rud-sh` [User Option]

standard-value
((funcall (function (closure (t) nil (concat R_PAPERSIZE=letter R_BATCHSAVE=’–no-save –no-restore’ R_BROWSER=xdg-open PAGER=/usr/bin/pager PERL=/usr/bin/perl R_UNZIPCMD=/usr/bin/unzip R_ZIPCMD=/usr/bin/zip R_PRINTCMD=/usr/bin/lpr LIBnn=lib AWK=/usr/bin/awk CC=’ccache gcc’ CFLAGS=’-ggdb -pipe -std=gnu99 -Wall -pedantic’ CXX=’ccache g++’ CXXFLAGS=’-ggdb -pipe -Wall -pedantic’ FC=’ccache gfortran’ F77=’ccache gfortran’ MAKE=’make -j4’ ./configure –prefix= rp-path-to-rud –enable-R-shlib –with-blas –with-lapack –with-readline –without-recommended-packages make make install)))))

custom-type
: (string)

custom-requests
: nil

variable-documentation
: Shell script to install R Under development.

5 Other symbolic expressions

```
(provide 'rp)
```

Index