Package 'rewind'

August 21, 2026

Type Package

Title Undo and Redo for 'Shiny' Applications

Version 0.2.0

Description Adds a user-facing undo and redo history to 'Shiny' applications. Application state, comprising registered inputs and optionally server-side reactive values, is captured as the user interacts with the application. Users may then step backwards and forwards through that history with the keyboard, with buttons, or by scrubbing a visual history rail. Rapid successive changes, such as dragging a slider, are coalesced into a single history entry, and related changes may be grouped explicitly into semantic steps.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1.0)

Imports htmltools, R6, shiny (>= 1.7.0), utils

Suggests shinytest2, testthat (>= 3.0.0), withr

URL https://github.com/tenmeh/rewind, https://tenmeh.github.io/rewind/

BugReports https://github.com/tenmeh/rewind/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Tanmay Chanda [aut, cre, cph]

Maintainer Tanmay Chanda <tanmaychanda96@gmail.com>

Repository CRAN

Date/Publication 2026-08-21 13:50:08 UTC

Contents

`rewind_buttons` *Undo and redo buttons*

Description

These two buttons connect to the history of the session. They become enabled and disabled as the history permits. You thus do not have to write an observer on the server.

Usage

```
rewind_buttons(undo_label = "Undo", redo_label = "Redo", class = NULL)
```

Arguments

`undo_label`, `redo_label`
: The button labels. Use `NULL` for a button with an icon only. The function then sets the accessible name (`aria-label`) to "Undo" or "Redo". A screen reader can thus announce the button correctly when it has no text.

`class`
: More CSS classes for the container element. The buttons have the classes `btn btn-default`. Bootstrap themes thus apply to them.

Value

A `htmltools::tagList()`.

Examples

```
rewind_buttons()
rewind_buttons(undo_label = NULL, redo_label = NULL)
```

rewind_dependency *The rewind HTML dependency*

Description

`rewind_enable()` adds this automatically. You thus rarely need this function. Use it when you want the assets before the server function runs. Use it also when you cannot use `insertUI()`.

Usage

```
rewind_dependency()
```

Value

An `htmltools::htmlDependency()`.

Examples

```
rewind_dependency()
```

rewind_disable *Fully disable undo/redo for a session*

Description

`rewind_pause()` stops capture for a short time. It needs a `rewind_resume()` call after it. This function is different. It removes everything that `rewind_enable()` made:

Usage

```
rewind_disable(session = shiny::getDefaultReactiveDomain())
```

Arguments

`session`	The Shiny session. The default is the current session.

Details

- it destroys each observer;
- it sets the buttons and the history rail in the browser to their empty, disabled condition;
- it returns the session to the condition before `rewind_enable()`.

Call `rewind_enable()` again to start a new history.

Use this function when the application permits undo and redo only in some conditions. An example is a user role that the application does not know at the start of the session.

Value

TRUE if the function disabled a session. FALSE if rewind was not enabled. The function returns the value invisibly.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()
    observeEvent(input$readonly_mode, {
      if (input$readonly_mode) rewind_disable()
    })
  }
}
```

---

rewind_enable *Enable undo and redo for a Shiny session*

Description

Call this once, near the top of your server function. After that, rewind records the session inputs as the user works. It also records the reactive values that you register with rewind_track(). The user can then move backwards and forwards through that history.

Usage

```
rewind_enable(
  session = shiny::getDefaultReactiveDomain(),
  inputs = NULL,
  exclude = NULL,
  depth = 50L,
  coalesce_ms = 400L,
  shortcuts = TRUE,
  verbose = FALSE
)
```

Arguments

session	The Shiny session. The default is the current session.
inputs	Character vector of the input IDs to capture. Use NULL (the default) to capture all permitted inputs.
exclude	Character vector of input IDs to skip. rewind applies this after inputs.
depth	The maximum number of history entries to keep. rewind removes the oldest entries first.

coalesce_ms	The quiet period in milliseconds before rewind writes a change to the history. Increase it to group more changes.
shortcuts	Set to TRUE to bind Ctrl/Cmd + Z and Ctrl/Cmd + Shift + Z (and Ctrl + Y) in the browser. The shortcuts do nothing while the user types in a text field. The text undo of the browser thus continues to work.
verbose	Set to TRUE to show messages about what rewind captures and restores. This is useful during development.

Value

The controller, invisibly. Most applications can ignore it.

What gets captured

By default rewind captures every input in the session. There are four exclusions. It is never useful to restore these:

- action buttons and links. Their value is a click counter.
- shiny::fileInput(). Its value points to a temporary file on the server. Shiny deletes that file at the next upload. An old snapshot would thus point to a file that does not exist.
- inputs with names that start with rewind_. These belong to this package.
- inputs with names that start with .. These are internal to Shiny.

Use inputs to give a list of the inputs to capture. This is usually better in a large application. Undo must move the controls that the user thinks of as filters. It must not move every other input on the page.

Grouping

Changes that occur within coalesce_ms of each other become one history entry. One drag of a slider is thus one undo step, not forty. Use rewind_step() to group changes yourself.

Modules

You can call this function inside a moduleServer(). rewind captures the inputs with their module-local names. These are the same names that input$ uses inside the module. rewind adds the namespace with session$ns() when it restores them.

All modules share session$userData. A second call to rewind_enable() thus uses the same history as the first call. It does not make a second history. Call the function once, at the position in the module tree that is best for your application.

Examples

```
if (interactive()) {
  library(shiny)

  ui <- fluidPage(
    rewind_buttons(),
    selectInput("species", "Species", c("setosa", "versicolor", "virginica")),
```

```
    sliderInput("n", "Rows", 1, 50, 10),
    tableOutput("tbl")
  )

  server <- function(input, output, session) {
    rewind_enable()

    output$tbl <- renderTable({
      head(iris[iris$Species == input$species, ], input$n)
    })
  }

  shinyApp(ui, server)
}
```

`rewind_history` *Inspect the history*

Description

`rewind_history()` gives one row for each entry that `rewind` keeps. `rewind_can_undo()` and `rewind_can_redo()` tell you if the position can change. All three functions depend on the history. You can thus use them in `render*()` and `observe()` to control your own UI.

Usage

```
rewind_history(session = shiny::getDefaultReactiveDomain())

rewind_can_undo(session = shiny::getDefaultReactiveDomain())

rewind_can_redo(session = shiny::getDefaultReactiveDomain())
```

Arguments

`session`	The Shiny session. The default is the current session.

Value

`rewind_history()` gives a data frame. It has the columns `index`, `label`, `time` and `current`. The other two functions give one logical value.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()
    output$steps <- renderTable(rewind_history())
```

```
  }
}
```

`rewind_pause` *Suspend and resume history capture*

Description

While capture is paused, the application applies changes as usual. But `rewind` does not record them. Use this around changes that your code makes, and that the user must not step back into. Examples are a saved session that you restore, or a URL bookmark that you apply.

Usage

```
rewind_pause(session = shiny::getDefaultReactiveDomain())

rewind_resume(session = shiny::getDefaultReactiveDomain())
```

Arguments

`session`	The Shiny session. The default is the current session.

Value

`TRUE`, invisibly.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()
    rewind_pause()
    # ... apply a saved bookmark here ...
    rewind_resume()
  }
}
```

`rewind_step` *Group several changes into one undo step*

Description

This function holds a block of code. Every change in that block becomes one history entry with the label that you give. Use it for buttons such as "reset all filters" or "apply preset". For these buttons, the standard time-based grouping can make several steps, or it can give a label that does not help the user.

Usage

```
rewind_step(
  expr,
  label = NULL,
  hold_ms = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`expr`	The code to run. This is usually a set of `update*Input()` calls, or assignments to tracked reactive values.
`label`	The label for the history entry.
`hold_ms`	The time in milliseconds to keep the entry open. The default is two times the `coalesce_ms` of the session. Increase it if a slow browser makes two steps from one block.
`session`	The Shiny session. The default is the current session.

Details

The block runs immediately. `rewind` writes the history entry after the changes return from the browser.

Value

The value of `expr`, invisibly.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()

    observeEvent(input$reset, {
```

```
      rewind_step(label = "Reset filters", {
        updateSelectInput(session, "region", selected = "All")
        updateSliderInput(session, "year", value = c(2000, 2026))
        updateCheckboxInput(session, "only_active", value = FALSE)
      })
    })
  }
}
```

`rewind_track` *Include server-side reactive values in the history*

Description

`rewind` captures inputs automatically. It cannot see the state that you keep in a `shiny::reactiveValues()` object. Register that object here. `rewind` then records the registered fields with the inputs, and writes them back at an undo.

Usage

```
rewind_track(
  values,
  fields = NULL,
  id = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`values`	A `shiny::reactiveValues()` object.
`fields`	Character vector of the field names to track. Use `NULL` for all the fields that exist when `rewind` takes the snapshot.
`id`	A name for this group. `rewind` uses it to keep the registered objects apart, and to label the history entries. The default is the name of the `values` argument at the call.
`session`	The Shiny session. The default is the current session.

Details

Register only the values that are true *state*. A derived value or a cached value does no harm, but it has no use. Do not register a value that an observer computes again immediately. The undo step then appears to do nothing.

Value

`TRUE`, invisibly.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()
    state <- reactiveValues(selected = character(), zoom = 1)
    rewind_track(state, fields = c("selected", "zoom"))
  }
}
```

`rewind_ui` *A scrubbable history rail*

Description

This function draws the history as a vertical list of steps. The newest step is at the end. The rail shows the current position. A click on a step moves to that step. The rail gets its data from the server. You thus do not have to write a `render` function.

Usage

```
rewind_ui(label = "History", max_height = "20rem", class = NULL)
```

Arguments

`label`	The heading above the rail. Use `NULL` for no heading.
`max_height`	The CSS height at which the rail starts to scroll.
`class`	More CSS classes for the container element.

Value

A `htmltools::tagList()`.

Examples

```
rewind_ui()
rewind_ui(label = "Steps", max_height = "12rem")
```

rewind_undo *Move through the history programmatically*

Description

The keyboard shortcuts and `rewind_buttons()` are sufficient for most applications. Use these functions to move through the history from your own controls.

Usage

```
rewind_undo(session = shiny::getDefaultReactiveDomain())

rewind_redo(session = shiny::getDefaultReactiveDomain())

rewind_jump(index, session = shiny::getDefaultReactiveDomain())

rewind_clear(session = shiny::getDefaultReactiveDomain())
```

Arguments

`session`	The Shiny session. The default is the current session.
`index`	The position to move to. The first position is 1. Position 1 holds the oldest entry that `rewind` keeps.

Value

`TRUE` if the position changed. If not, `FALSE`. The functions return the value invisibly.

Examples

```
if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    rewind_enable()
    observeEvent(input$my_back_button, rewind_undo())
  }
}
```

Index