EBook Gratis

# APRENDIZAJE shiny

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#shiny

# Tabla de contenido

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: shiny

It is an unofficial and free shiny ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official shiny.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con brillante

## Observaciones

Esta sección proporciona una descripción general de qué es el brillo y por qué un desarrollador puede querer usarlo.

También debe mencionar los temas grandes dentro de los brillantes, y vincular a los temas relacionados. Dado que la Documentación para brillos es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Examples

### Instalación o configuración

Shiny puede ejecutarse como una aplicación independiente en su computadora local, en un servidor que puede proporcionar aplicaciones brillantes a múltiples usuarios (usando el servidor de shiny), o en shinyapps.io .

1. **Instalar Shiny en una computadora local:** en R / RStudio, ejecute `install.packages("shiny")` si está instalando desde CRAN, o `devtools::install_github("rstudio/shiny")` si está instalando desde el repositorio RStudio Github. El repositorio Github aloja una versión de desarrollo de Shiny que posiblemente tenga más funciones en comparación con la versión CRAN, pero también puede ser inestable.

### ¿Cuándo usaría el brillo?

1. Tengo algunos análisis de datos realizados sobre algunos datos y tengo muchas personas 'no codificadas' en el equipo, que tienen datos similares a los míos y tienen requisitos de análisis similares. En tales casos, puedo crear una aplicación web con brillos, que toma archivos de datos de entrada específicos del usuario y genera análisis.
2. Necesito compartir datos analizados o diagramas relevantes con otros en el equipo. Las aplicaciones web brillantes pueden ser útiles en tales situaciones.
3. No tengo experiencia significativa con la programación de aplicaciones web, pero necesito ensamblar rápidamente una interfaz simple. Brillante para el rescate con una interfaz de usuario sencilla y elementos de servidor y codificación mínima.
4. Los elementos interactivos permiten a los usuarios explorar qué elemento de los datos es relevante para ellos. Por ejemplo, puede tener datos cargados para toda la compañía, pero tiene un menú desplegable por departamento como "Ventas", "Producción", "Finanzas" que puede resumir los datos de la forma en que los usuarios desean verlos. La alternativa sería producir un enorme paquete de informes con análisis para cada departamento, pero solo leen su capítulo y el total.

## Aplicación simple

Cada aplicación `shiny` contiene dos partes: una definición de interfaz de usuario ( `UI` ) y un script de `server` ( `server` ). Este ejemplo muestra cómo se puede imprimir "Hola mundo" desde la interfaz de usuario o desde el servidor.

**IU.R**

En la interfaz de usuario puede colocar algunos objetos de vista (div, entradas, botones, etc.).

```
library(shiny)

# Define UI for application print "Hello world"
shinyUI(

  # Create bootstrap page
  fluidPage(

    # Paragraph "Hello world"
    p("Hello world"),

    # Create button to print "Hello world" from server
    actionButton(inputId = "Print_Hello", label = "Print_Hello World"),

    # Create position for server side text
    textOutput("Server_Hello")

  )
)
```

**Servidor.R**

En el script del servidor puede definir métodos que manipulan datos o escuchan acciones.

```
# Define server logic required to print "Hello World" when button is clicked
shinyServer(function(input, output) {

  # Create action when actionButton is clicked
  observeEvent(input$Print_Hello,{

    # Change text of Server_Hello
    output$Server_Hello = renderText("Hello world from server side")
  })


})
```

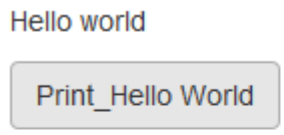
**¿Como correr?**

Puede ejecutar su aplicación de varias maneras:

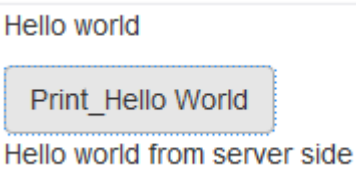
1. Cree dos archivos diferentes y colóquelos en un directorio, luego use `runApp('your dir path')`
2. Puede definir dos variables (ui y servidor, por ejemplo) y luego usar `shinyApp(ui,server)` para ejecutar su aplicación

**Resultado**

En este ejemplo verá un texto y un botón:




Y después de hacer clic en el botón el servidor responde:




## Incluyendo parcelas

La forma más sencilla de incluir gráficos en su shinyApp es usar `plotOutput` en la interfaz de `renderPlot` y `renderPlot` en el servidor. Esto funcionará con gráficos base así como con `ggPlot` s

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  plotOutput('myPlot'),
  plotOutput('myGgPlot')
)

server <- function(input, output, session){
  output$myPlot = renderPlot({
    hist(rnorm(1000))
  })
  output$myGgPlot <- renderPlot({
    ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) + geom_point()
  })
}

shinyApp(ui, server)
```

## Incluyendo mesas

Las tablas se incluyen más fácilmente con el paquete DT , que es una interfaz R para la biblioteca de JavaScript DataTables.

```
library(shiny)
library(DT)
```

```
ui <- fluidPage(
  dataTableOutput('myTable')
)

server <- function(input, output, session){
  output$myTable <- renderDataTable({
    datatable(iris)
  })
}

shinyApp(ui, server)
```

Lea Empezando con brillante en línea: https://riptutorial.com/es/shiny/topic/2667/empezando-con-brillante

# Capítulo 2: API de Javascript

## Sintaxis

- sesión $ sendCustomMessage ( *nombre* , *lista de parámetros* )
- Shiny.addCustomMessageHandler ( *nombre* , *función JS que acepta la lista de parámetros* )
- Shiny.onInputChange ( *nombre* , *valor* )

## Examples

### Enviando datos del servidor al cliente

En muchos casos, deseará enviar datos desde el servidor R al cliente JS. Aquí hay un ejemplo muy simple:

```
library(shiny)
runApp(
  list(
    ui = fluidPage(
      tags$script(
        "Shiny.addCustomMessageHandler('message', function(params) { alert(params); });"
      ),
      actionButton("btn","Press Me")
    ),
    server = function(input, output, session) {
      observeEvent(input$btn,{
        randomNumber <- runif(1,0,100)
        session$sendCustomMessage("message",list(paste0(randomNumber," is a random number!")))
      })
    }
  )
)
```

Los caballos de trabajo aquí son la función `session$sendCustomMessage` en `R` y la función `Shiny.addCustomMessageHandler` en `javascript` .

La función `session$sendCustomMessage` permite enviar parámetros desde `R` a una función `javascript` , y `Shiny.addCustomMessageHandler` permite definir la función `javascript` que acepta los parámetros de `R`

Nota: las listas se convierten a JSON cuando se pasan de `R` a `javascript`

### Enviando datos de cliente a servidor

En algunos casos, deseará enviar datos desde el cliente JS al servidor R. Aquí hay un ejemplo básico utilizando la función `Shiny.onInputChange` de javascript:

```
library(shiny)
runApp(
```

```
  list(
    ui = fluidPage(
      # create password input
      HTML('<input type="password" id="passwordInput">'),
      # use jquery to write function that sends value to
      # server when changed
      tags$script(
        '$("#passwordInput").on("change",function() {
          Shiny.onInputChange("myInput",this.value);
        })'
      ),
      # show password
      verbatimTextOutput("test")
    ),
    server = function(input, output, session) {
      # read in then show password
      output$test <- renderPrint(
        input$myInput
      )
    }
  )
)
```

Aquí creamos una entrada de contraseña con id `passwordInput` . Agregamos una función de Javascript en la interfaz de usuario que reacciona a los cambios en `passwordInput` y envía el valor al servidor usando `Shiny.onInputChange` .

`Shiny.onInputChange` toma dos parámetros, un nombre para la `input$*name*` , más un valor para la `input$*name*`

Luego puede usar la `input$*name*` como cualquier otra entrada brillante.

Lea API de Javascript en línea: https://riptutorial.com/es/shiny/topic/3149/api-de-javascript

# Capítulo 3: Cómo escribir aplicaciones de MCVE (ejemplo mínimo, completo y verificable) de Shiny

## Introducción

Si tiene problemas con sus aplicaciones Shiny, es una buena práctica crear una aplicación que ilustre su punto. Esta aplicación debe ser lo más simple posible, sin dejar de reflejar su problema. Esto significa usar conjuntos de datos simples, nombres autoexplicativos (especialmente para las ID de E / S) y reemplazar los gráficos por otros más simples.

También es recomendable crear su MCVE de manera que se requiera la menor cantidad de bibliotecas no estándar.

## Examples

### Estructura basica

Los MCVE deben iniciar la aplicación Shiny cuando se copian en la consola. Una forma fácil de hacer esto es usar la función `shinyApp` . Por ejemplo:

> ¿Por qué mi casilla de verificación no responde?

```
library(shiny)

ui <- fluidPage(
  checkboxInput('checkbox', 'click me'),
  verbatimTextOutput('text')
)

server <- function(input, output, session){
  output$text <- renderText({
    isolate(input$checkbox)
  })
}

shinyApp(ui, server)
```

Alternativamente, tampoco puede asignar variables a la `ui server` y al `server` .

```
library(shiny)

shinyApp(
  fluidPage(
    checkboxInput('checkbox', 'click me'),
    verbatimTextOutput('text')
  ),
```

```
  function(input, output, session){
    output$text <- renderText({
      isolate(input$checkbox)
    })
  }
)

shinyApp(ui, server)
```

### Evitar detalles innecesarios

En la práctica, las aplicaciones brillantes a menudo son muy complicadas y están llenas de características que se han desarrollado a lo largo del tiempo. La mayoría de las veces, esos detalles adicionales no son necesarios para reproducir su problema. Es mejor si omite tales detalles al escribir MCVE.

# INCORRECTO

> ¿Por qué no se muestra mi trama?

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  plotOutput('plot')
)

server <- function(input, output, session){
  df <- data.frame(treatment = rep(letters[1:3], times = 3),
                   context = rep(LETTERS[1:3], each = 3),
                   effect = runif(9,0,1))
  df$treat.con <- paste(df$treatment,df$context, sep = ".")
  df$treat.con <- reorder(df$treat.con, -df$effect, )
  output$plot = renderPlot({
    myPlot <- ggplot(df, aes(x = treat.con, y = effect)) +
       geom_point() +
       facet_wrap(~context,
                  scales = "free_x",
                  ncol = 1)
  })
}

shinyApp(ui, server)
```

# CORRECTO

> ¿Por qué mi trama no se muestra?

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
```

```
  plotOutput('plot')
)

server <- function(input, output, session){
  output$plot = renderPlot({
    myPlot <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
  })
}

shinyApp(ui, server)
```

Lea Cómo escribir aplicaciones de MCVE (ejemplo mínimo, completo y verificable) de Shiny en línea: https://riptutorial.com/es/shiny/topic/10653/como-escribir-aplicaciones-de-mcve--ejemplo-minimo--completo-y-verificable--de-shiny

# Capítulo 4: reactivo, reactivoValor y eventoReactivo, observe y observe Evento en Shiny

## Introducción

**reactivo, valor reactivo y eventoReactivo** son varios tipos de expresiones reactivas en Shiny. Producen una salida que se puede usar como entrada en otras expresiones, que a su vez dependerá de la expresión reactiva.

**observar y observar los eventos** son similares a las expresiones reactivas. La gran diferencia es que los observadores no producen ningún resultado y, por lo tanto, solo son útiles para sus efectos secundarios.

Ejemplos de su uso se dan en este documento.

## Examples

### reactivo

Se puede usar un reactivo para hacer que la salida dependa de otra expresión. En el siguiente ejemplo, el elemento de salida de texto $ depende de text_reactive, que a su vez depende de la entrada $ user_text. Cuando la entrada $ user_text cambia, la salida $ text element y text_reactive se **invalidan.** Se recalculan según el nuevo valor para la entrada $ user_text.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # reactive expression
  text_reactive <- reactive({
    input$user_text
  })

  # text output
```

```
  output$text <- renderText({
    text_reactive()
  })
}

shinyApp(ui = ui, server = server)
```

### eventoReactivo

Los eventosReactivos son similares a los reactivos, se construyen de la siguiente manera:

```
eventReactive( event {
code to run
})
```

eventReactives no dependen de todas las expresiones reactivas de su cuerpo (' *código para ejecutar* ' en el fragmento de *código* anterior). En su lugar, solo dependen de las expresiones especificadas en la sección de *eventos* .

En el siguiente ejemplo, hemos agregado un botón de envío y hemos creado un eventoReactive. Cuando la entrada $ user_text cambia, el evento de reacción no se invalida, ya que el evento de respuesta solo depende de la entrada de acción de botón $ enviar. Cada vez que se presiona ese botón, text_reactive y luego se imprime $ text se invalida, y se recalulará según la entrada actualizada $ user_text.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example eventReactive"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),

    # submit button
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # reactive expression
  text_reactive <- eventReactive( input$submit, {
    input$user_text
  })

  # text output
  output$text <- renderText({
    text_reactive()
  })
}
```

```
shinyApp(ui = ui, server = server)
```

## valores reactivos

reactiveValues se puede usar para almacenar objetos, a los que otras expresiones pueden llevar una dependencia.

En el siguiente ejemplo, un objeto reactiveValues se inicializa con el valor "Aún no se ha enviado ningún texto". Se crea un observador independiente para actualizar el objeto reactiveValues cada vez que se presiona el botón de envío. Tenga en cuenta que el valor reactivo en sí mismo no depende de las expresiones de su cuerpo.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactiveValues"),

  mainPanel(

    # input field
    textInput("user_text", label = "Enter some text:", placeholder = "Please enter some
text."),
    actionButton("submit", label = "Submit"),

    # display text output
    textOutput("text"))
)

server <- function(input, output) {

  # observe event for updating the reactiveValues
  observeEvent(input$submit,
               {
    text_reactive$text <- input$user_text
  })

  # reactiveValues
  text_reactive <- reactiveValues(
    text = "No text has been submitted yet."
  )

  # text output
  output$text <- renderText({
    text_reactive$text
  })
}

shinyApp(ui = ui, server = server)
```

## observar

Se puede utilizar un objeto observeEvent para desencadenar un fragmento de código cuando se produce un evento determinado. Se construye como:

```
observeEvent( event {
code to run
})
```

El observEvent solo dependerá de la sección de *'evento'* en el pequeño fragmento de código de arriba. No dependerá de nada en la parte del ' *código para ejecutar* '. Una implementación de ejemplo se puede encontrar a continuación:

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(

    # action buttons
    actionButton("button1","Button 1"),
    actionButton("button2","Button 2")
  )
)

server <- function(input, output) {

  # observe button 1 press.
  observeEvent(input$button1, {
    # The observeEvent takes no dependency on button 2, even though we refer to the input in
the following line.
    input$button2
    showModal(modalDialog(
      title = "Button pressed",
      "You pressed one of the buttons!"
    ))
  })
}

shinyApp(ui = ui, server = server)
```

## observar

Una expresión de observación se activa cada vez que cambia una de sus entradas. La principal diferencia con respecto a una expresión reactiva es que no produce resultados, y solo debe usarse para sus efectos secundarios (como la modificación de un objeto de valores reactivos o la activación de una ventana emergente).

Además, tenga en cuenta que observar no ignora los valores NULL, por lo que se activará incluso si sus entradas siguen siendo NULL. observeEvent de forma predeterminada, ignora NULL, ya que casi siempre es deseable.

```
library(shiny)

ui <- fluidPage(
  headerPanel("Example reactive"),

  mainPanel(
```

```
    # action buttons
    actionButton("button1","Button 1"),
    actionButton("button2","Button 2")
  )
)

server <- function(input, output) {

  # observe button 1 press.
  observe({
    input$button1
    input$button2
    showModal(modalDialog(
      title = "Button pressed",
      "You pressed one of the buttons!"
    ))
  })
}

shinyApp(ui = ui, server = server)
```

Lea reactivo, reactivoValor y eventoReactivo, observe y observe Evento en Shiny en línea: https://riptutorial.com/es/shiny/topic/10787/reactivo--reactivovalor-y-eventoreactivo--observe-y-observe-evento-en-shiny

# Capítulo 5: Subir datos a brillosos

## Examples

### Suba archivos .RData a brilloso con fileInput ()

El ejemplo te permite subir archivos .RData. El enfoque con `load` y `get` permite asignar los datos cargados a un nombre de variable de su elección. En el caso de que el ejemplo sea "independiente", inserté la sección superior que almacena dos vectores en el disco para cargarlos y trazarlos más tarde.

```
library(shiny)

# Define two datasets and store them to disk
x <- rnorm(100)
save(x, file = "x.RData")
rm(x)
y <- rnorm(100, mean = 2)
save(y, file = "y.RData")
rm(y)

# Define UI
ui <- shinyUI(fluidPage(
  titlePanel(".RData File Upload Test"),
  mainPanel(
    fileInput("file", label = ""),
    actionButton(inputId="plot","Plot"),
    plotOutput("hist"))
  )
)

# Define server logic
server <- shinyServer(function(input, output) {

  observeEvent(input$plot,{
    if ( is.null(input$file)) return(NULL)
    inFile <- input$file
    file <- inFile$datapath
    # load the file into new environment and get it from there
    e = new.env()
    name <- load(file, envir = e)
    data <- e[[name]]

    # Plot the data
    output$hist <- renderPlot({
      hist(data)
    })
  })
})

# Run the application
shinyApp(ui = ui, server = server)
```

### Subiendo archivos csv a Shiny

También es posible que un usuario suba los csv a su aplicación Shiny. El siguiente código muestra un pequeño ejemplo de cómo se puede lograr esto. También incluye una entrada de radioButton para que el usuario pueda elegir de forma interactiva el separador que se utilizará.

```
library(shiny)
library(DT)

# Define UI
ui <- shinyUI(fluidPage(

  fileInput('target_upload', 'Choose file to upload',
            accept = c(
              'text/csv',
              'text/comma-separated-values',
              '.csv'
            )),
  radioButtons("separator","Separator: ",choices = c(";",",","","":"), selected=";",inline=TRUE),
  DT::dataTableOutput("sample_table")
)
)

# Define server logic
server <- shinyServer(function(input, output) {

  df_products_upload <- reactive({
    inFile <- input$target_upload
    if (is.null(inFile))
      return(NULL)
    df <- read.csv(inFile$datapath, header = TRUE,sep = input$separator)
    return(df)
  })

  output$sample_table<- DT::renderDataTable({
    df <- df_products_upload()
    DT::datatable(df)
  })

}
)

# Run the application
shinyApp(ui = ui, server = server)
```

Lea Subir datos a brillosos en línea: https://riptutorial.com/es/shiny/topic/7576/subir-datos-a-brillosos

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con brillante | Batanichek, Bogdan Rau, chrki, Community, Florian, Gregor de Cillia, jsb, Mathias711, micstr, sigmabeta |
| 2 | API de Javascript | Carl, Tomás Barcellos |
| 3 | Cómo escribir aplicaciones de MCVE (ejemplo mínimo, completo y verificable) de Shiny | Florian, Gregor de Cillia |
| 4 | reactivo, reactivoValor y eventoReactivo, observe y observe Evento en Shiny | Florian |
| 5 | Subir datos a brillosos | Florian, symbolrush |