Software Carpentry

Plantilla de proyecto en R: carpetas, rutas y control del sistema

Dante Conti, Sergi Ramirez, (c) IDEAI

2025-09-21

Table of contents

1 Objetivo

Este notebook muestra **cómo organizar un proyecto de R** y controlar operaciones de sistema:

- Estructura recomendada de carpetas: `syntax/`, `input/`, `output/`, `data/`, `temp/`, `logs/`.
- Manejo de **rutas relativas** con `{here}`.
- Mensajes y nombres de archivo dinámicos con `{glue}`.
- Crear y buscar ficheros (`dir.create()`, `file.path()`, `list.files()`, `fs::dir_create()`…).
- Redirigir salida con `sink()`.
- Guardar gráficos con `pdf()` y `png()`.
- Mini *pipeline* de ejemplo (leer → procesar → guardar).

  Consejo: evita `setwd()` y usa rutas relativas con `{here}` para que tu proyecto sea 100% reproducible.

2 Paquetes y opciones

```
# Paquetes base y útiles:
packs <- c("here", "glue", "fs", "readr", "dplyr", "ggplot2")
to_install <- setdiff(packs, rownames(installed.packages()))
if (length(to_install)) install.packages(to_install, quiet = TRUE)

library(here)    # Rutas relativas desde la raíz del proyecto
library(glue)    # Strings con llaves {var}
library(fs)      # Operaciones de sistema "friendly"
library(readr)   # Lectura/escritura rápida
library(dplyr)   # Manipulación de datos
library(ggplot2) # Gráficos

# Opciones útiles
options(
  scipen = 999,   # menos notación científica
  digits = 4
)

# Mostrar dónde cree {here} que está la raíz del proyecto
here()
```

¿Cómo define `{here}` la raíz?

- Buscar archivo(s) "ancla" (`.Rproj`, `.here`, `DESCRIPTION`, `git/`, etc).
- Si no encuentra, puede crear un archivo vacío llamado `.here` en la carpeta raíz del proyecto.

```
# Crea un archivo marcador para que {here} sepa que esta carpeta es la raíz:
file_create(".here")
```

3 Estructura del proyecto

La estructura propuesta es la siguiente:

```
project/
  syntax/        # scripts R (funciones, notebooks, etc.)
  input/         # insumos externos (CSV, XLSX, etc.) SOLO LECTURA
  data/          # datos intermedios limpios/parquet/rds
  output/        # resultados finales (tablas/figuras/listados)
  temp/          # temporales desechables
  logs/          # logs de ejecución
  README.md
  .here          # marca la raíz del proyecto p/ {here}
```

Para crearlo, podemos hacerlo de la siguiente forma:

```
# Crear la estructura de carpetas si no existe:
dirs <- c("syntax", "input", "data", "output", "temp", "logs")
dir_create(path = here(dirs))
dir_ls(here(), type = "directory")
```

4 Estructura del proyecto

Usa `here("carpeta", "sub", "archivo.ext")` para **rutas portables**:

```
# Construir rutas de forma segura:
ruta_input  <- here("input", "ventas_2025.csv")
ruta_data   <- here("data",  "ventas_limpio.rds")
ruta_salida <- here("output","resumen_ventas.csv")

ruta_input
```

```r
ruta_data
ruta_salida

# Con base R: file.path() también es portable
file.path("input", "ventas_2025.csv")
```

Con `{glue}` puedes crear nombres dinámicos:

```r
anio <- 2025; mes <- 9
nombre_csv <- glue("ventas_{anio}-{sprintf('%02d', mes)}.csv")
here("input", nombre_csv)
```

5 Crear y escribir archivos

```r
# Datos de ejemplo:
df <- tibble::tibble(
  id = 1:5,
  fecha = as.Date("2025-09-01") + 0:4,
  ventas = c(100, 80, 95, 120, 110)
)

# Guardar como CSV en output/
write_csv(df, here("output", "tabla_ejemplo.csv"))

# Guardar como RDS en data/
saveRDS(df, here("data", "tabla_ejemplo.rds"))
```

6 Búsqueda de ficheros

`list.files()` (base) y `fs::dir_ls()` (recursivo, con *globbing*):

```r
# Listado simple
list.files(here("output"))

# Listado recursivo con patrón:
dir_ls(here(), recurse = TRUE, glob = "output/*.csv")
```

```r
# Buscar por ext. en múltiples carpetas:
dir_ls(here(c("input","data","output")), recurse = TRUE,
       regexp = "\\.(csv|rds)$")
```

7 Redirección de salida con `sink()`

```r
log_path <- here("logs", glue("log_{format(Sys.time(), '%Y%m%d_%H%M%S')}.txt"))

sink(log_path, split = TRUE)      # split=TRUE => también muestra en consola
cat("=== INICIO ===\n")
print(sessionInfo())
cat("Una línea cualquiera\n")
sink()  # IMPORTANTÍSIMO: cerrar el sink

# Revisa el contenido del log:
readLines(log_path, n = 8)
```

Cierra siempre el `sink()` con `sink()` (sin argumentos) o usa `on.exit(sink())` dentro de una función para no "bloquear" la consola.

8 Dispositivos gráficos: `pdf()` y `png()`

Puedes abrir un **dispositivo** gráfico, dibujar y cerrado con `dev.off()`.

```r
pdf(here("output", "grafico_demo.pdf"), width = 7, height = 5)
plot(cars, main = "Gráfico base R - cars")
dev.off()

# PNG con resolución
png(here("output", "grafico_demo.png"), width = 1200, height = 900, res = 150)
plot(pressure, main = "Gráfico base R - pressure")
dev.off()
```

Con **ggplot2**:

```r
p <- ggplot(mtcars, aes(disp, mpg)) + geom_point() +
  labs(title = "Relación cilindrada vs. mpg")

# Guardar directamente
ggsave(filename = here("output", "mtcars_disp_mpg.png"), plot = p,
       width = 7, height = 5, dpi = 150)

# También PDF
ggsave(filename = here("output", "mtcars_disp_mpg.pdf"), plot = p,
       width = 7, height = 5)
```

9 Buenas prácticas con {here}

- Coloca un archivo `.here` o un `.Rproj` en la raíz
- **Nunca** uses `setwd()` dentro de scripts reutilizables.
- Escribe funciones que reciban rutas **como argumento** o que construyan rutas con `here()`.

```r
# Función ejemplo usando here()
lee_input <- function(nombre) {
  readr::read_csv(here("input", nombre), show_col_types = FALSE)
}

# Uso:
# df <- lee_input("ventas_2025.csv")
```

10 Mensajes y nombres con {glue}

```r
# Glue para strings explicativos
archivo <- "ventas_2025.csv"
mensaje <- glue("Leyendo el archivo '{archivo}' desde {here('input')}")
mensaje
```

`glue()` evalúa expresiones dentro de `{}`:

```r
clientes <- 1250
glue("Este mes se han registrado {clientes} clientes (Δ = {clientes - 1200}).")
```

11 Mini pipeline: leer → procesar → guardar

Ejemplo autocontenido que crea un CSV de entrada, lo procesa y guarda resultados.

```
# 1) Crear un CSV de ejemplo en input/
dir_create(here("input"))
toy <- tibble::tibble(
  id = 1:10,
  fecha = as.Date("2025-09-01") + 0:9,
  ventas = sample(80:150, 10, replace = TRUE)
)
write_csv(toy, here("input", "toy_ventas.csv"))

# 2) Leer, procesar y registrar
log_path <- here("logs", "mini_pipeline.log")
sink(log_path, split = TRUE)
cat("== MINI PIPELINE ==\n")

raw <- read_csv(here("input", "toy_ventas.csv"), show_col_types = FALSE)
cat(glue("Leídas {nrow(raw)} filas.\n"))

proc <- raw |>
  mutate(
    semana = format(fecha, "%Y-%W"),
    ventas_norm = scale(ventas)[,1]
  ) |>
  group_by(semana) |>
  summarise(ventas_media = mean(ventas), .groups = "drop")

cat(glue("Semanas agregadas: {nrow(proc)}\n"))

# 3) Guardar resultados
write_csv(proc, here("output", "resumen_semanal.csv"))
saveRDS(proc, here("data", "resumen_semanal.rds"))
cat("Archivos guardados en output/ y data/\n")
sink()

# 4) Graficar y guardar
p <- ggplot(raw, aes(fecha, ventas)) + geom_line() +
  labs(title = "Ventas diarias (toy)", x = "Fecha", y = "Ventas")
ggsave(here("output", "ventas_toy.png"), plot = p, width = 7,
       height = 5, dpi = 150)
```

12 Utilidades (helpers) para tus scripts de `syntax/`

```r
# Guardar en syntax/helpers.R y luego source("syntax/helpers.R") si quieres

init_log <- function(prefix = "run") {
  dir_create(here("logs"))
  path <- here("logs",
               glue("{prefix}_{format(Sys.time(), '%Y%m%d_%H%M%S')}.log"))
  sink(path, split = TRUE)
  cat(glue("[{Sys.time()}] INICIO\n"))
  return(path)
}

close_log <- function() {
  cat(glue("[{Sys.time()}] FIN\n"))
  sink()
}

safe_dir <- function(...) {
  # Crea una ruta y la carpeta si no existe
  path <- here(...)
  dir_create(dirname(path))
  return(path)
}

save_table <- function(df, ..., name, ext = "csv") {
  # Guarda tabla df en output/ con nombre dinámico
  base <- glue("{name}.{ext}")
  path <- safe_dir("output", base)
  if (ext == "csv") readr::write_csv(df, path)
  if (ext == "rds") saveRDS(df, sub("\\.csv$", ".rds", path))
  invisible(path)
}
```

13 Pautas de versión y limpieza

- Todo lo que **no** sea fuente, mételo bajo control (ej: borrar `/temp/` al finalizar).
- Usa `git` para versionar scripts y notebooks.
- Separa **lectura** (`input/`) de **resultados** (`output/`) y **datos de trabajo** (`data/`).

```r
# Limpieza de temporales
if (dir_exists(here("temp"))) {
  file_delete(dir_ls(here("temp"), recurse = TRUE, type = "file"))
}
```

14 Apéndice: alternativas útiles

- `fs::file_copy()`, `fs::file_move()`, `fs::file_delete()` para copiar/mover/borrar.
- `Sys.getenv("VAR")` para leer variables de entorno.
- `withr::with_dir()` para ejecutar código en otra dir sin cambiar tu wd global.

```r
# Copiar un archivo de ejemplo
fs::file_copy(here("output", "tabla_ejemplo.csv"),
              here("temp", "copia_tabla.csv"),
              overwrite = TRUE)

# Variables de entorno
Sys.getenv("HOME")
```

15 Session info

```r
sessionInfo()
```