Tema 4:

Uso de archivos externos

4.1. Función scan

Una de las funciones más versátiles, y por tanto más complejas, para leer datos de un archivo es la función **scan**, que lee datos desde un archivo de texto o de la entrada estándar. Los argumentos son los siguientes:

- **file.** Es una cadena de caracteres que especifica un archivo. Si el nombre es relativo, comenzará en el directorio de trabajo. Debe tener en cuenta que el separador entre directorios es **/** y si desea utilizar el símbolo **** debe escribirlo doble, ****, ya que este símbolo se utiliza para definir caracteres especiales. Si especifica la cadena vacía, los datos se leen desde la entrada estándar, esto es, el teclado y el final de la entrada de datos se realiza mediante una línea en blanco o con el símbolo de fin de archivo (Ctrl+Z en Windows).
- **what**. Es un vector de modo numérico, carácter o complejo o una lista de vectores de dichos tipos, de tal modo que los datos del archivo se interpretan como de ese tipo. Si no se indica nada los datos se consideran numéricos. Si es una lista, se considera que cada registro tiene **length(what)** campos y el modo de cada uno es el correspondiente en la lista.
- %Si \textbf{widths} es un vector de longitud superior a uno, \textbf{what} debe ser una lista de la misma longitud.
- **nmax**. Es el máximo número de datos a leer o si **what** es una lista, el máximo número de registros. Si se omite se lee hasta el final del archivo.
- **n**. Es el máximo número de datos a leer. Si se omite no existe límite.
- **sep**. Indica el carácter utilizado como separador entre campos. Puede utilizar el tabulador, **\t,** o el cambio de línea, **\n**. El valor predeterminado es uno o más blancos, o cambios de línea.
- **quote**. Una cadena de caracteres que corresponde a los caracteres que sirven para delimitar cadenas de caracteres.
- **dec**. El separador decimal.
- **skip**. Número de líneas que deben saltarse al inicio del archivo antes de comenzar a leer los datos.
- **nlines**. Número máximo de líneas para leer.
- **na.strings**. Caracteres que deben ser sustituidos, tras su lectura, por NA.
- **flush**. Si el valor es TRUE, una vez leídos el número de campos indicados, se descarta el resto de la línea.
- **fill**. Si el valor es TRUE, si una línea contiene menos datos de los esperados se rellenan con vacíos.
- **strip.white**. Vector de valores lógicos correspondientes a los elementos de **what** para indicar si deben eliminarse los espacios blancos al inicio y final de cada campo. En campos numéricos, siempre se eliminan.
- **quiet**. Valor lógico que indica si es FALSE, que es el valor predeterminado, que imprima el número de campos leídos.
- **blank.lines.skip.** Valor lógico que indica si es TRUE que se ignoren las líneas en blanco.
- **multi.line**. Valor lógico que se usa sólamente si **what** es una lista y que indica si es FALSE que una línea lógica no puede ocupar más de una línea física.
- **comment.char**. Un carácter que indica que el resto de la línea es un comentario. El valor predeterminado es la cadena vacía, lo que indica que no hay comentarios.

El resultado es un vector o una lista si se utiliza **what** y un vector numérico en caso contrario. Como regla general, debe tener en cuenta que, lo más cómodo, es preparar los datos para que su lectura no resulte complicada.

Si en un campo numérico se encuentra la palabra **NA** se entiende que es un valor no disponible.

Hay que destacar que cada vez que se lee un registro debe hacerse una llamada a la asignación de memoria para hacer sitio. Si se sabe previamente el tamaño de los vectores que se van a leer y se asigna este tamaño, se ahorra gran cantidad de memoria, lo que permite leer archivos mayores.

Para leer datos *bien preparados* la función **read.table** puede ser más conveniente.

Cuando el nombre de un archivo se da como un camino relativo y no absoluto, el nombre completo se resuelve completándolo desde el directorio de trabajo. Consúltelo con **getwd** --- usualmente se encuentra en \textbf{C:/Archivos de programa/rw2000/bin}--- o cámbielo con la función **setwd**.

Ejemplos

Peso <- scan() lee datos del teclado, hasta que se deje una línea en blanco.

Nombres <- scan("personas.txt", what="") Lee un vector de cadenas de caracteres del archivo "personas.txt"

Si desea leer un archivo y que los datos correspondan a una matriz puede hacerlo combinando la función **matrix** con **scan**. Por ejemplo, para los datos incluidos en el archivo *datos.txt*, la orden de lectura sería

```
> matrix(scan("c:/datos/datos.txt"),ncol=3,byrow=T)
[,1] [,2] [,3]
[1,] 77 1.63 23
[2,] 58 1.63 23
[3,] 89 1.85 26
[4,] 55 1.62 23
[5,] 47 1.60 26
[6,] 60 1.63 26
[7,] 54 1.70 22
[8,] 58 1.65 23
[9,] 75 1.78 26
[10,] 65 1.70 24
[11,] 82 1.77 28
[12,] 85 1.83 42
[13,] 75 1.74 25
[14,] 65 1.65 26
```

Si quiere dar nombre a las columnas puede hacerlo asignando los valores a una lista, **nombres**, y luego pasándola como parámetro a la función **matrix**.

```
> nombres <-list(c(),c("Peso","Altura","Edad"))
> matrix(scan("c:/datos/datos.txt"),ncol=3,dimnames=nombres, byrow=T)
Peso Altura Edad
[1,] 77 1.63 23
[2,] 58 1.63 23
[3,] 89 1.85 26
[4,] 55 1.62 23
[5,] 47 1.60 26
[6,] 60 1.63 26
[7,] 54 1.70 22
[8,] 58 1.65 23
[9,] 75 1.78 26
[10,] 65 1.70 24
[11,] 82 1.77 28
[12,] 85 1.83 42
[13,] 75 1.74 25
[14,] 65 1.65 26
```

También podría haberlo escrito directamente en una sola orden, aunque del modo anterior queda más claro.

```
> matrix(scan("c:/datos/datos.txt"),ncol=3,
dimnames=list(c(),c("Peso","Altura","Edad")), byrow=T)
Peso Altura Edad
[1,] 77 1.63 23
[2,] 58 1.63 23
[3,] 89 1.85 26
[4,] 55 1.62 23
[5,] 47 1.60 26
[6,] 60 1.63 26
[7,] 54 1.70 22
[8,] 58 1.65 23
[9,] 75 1.78 26
[10,] 65 1.70 24
[11,] 82 1.77 28
[12,] 85 1.83 42
[13,] 75 1.74 25
[14,] 65 1.65 26
```

4.1.1. Función readLines

Para leer un archivo por líneas completas puede utilizar la función **readLines** cuyos argumentos son

- **con**. El nombre del archivo
- **n**. Un entero que indica el máximo número de líneas para leer. Si se indica un valor negativo se leerá hasta alcanzar el final del archivo.
- **ok**. Valor lógico que indica si se espera alcanzar el final del archivo antes de leer

el número de líneas especificado como máximo, si no es así se genera un error.

El resultado es un vector de tipo carácter cuyos elementos son las líneas leídas. A continuación escribimos cuatro líneas en el archivo **prueba.txt** mediante la función **cat**. Los cambios de línea vienen marcados por la secuencia **\n**. Seguidamente leemos el archivo por líneas.

```
> cat("Hola", 12, 13, "Soy\n", "yo\n", "y\n", "te espero\n", file="prueba.txt")
> readLines("prueba.txt", n=-1)
[1] "Hola 12 13 Soy"
[2] " yo"
[3] " y"
[4] " te espero"
```

4.1.2. Función writeLines

Esta función escribe líneas de texto en un archivo. Sus argumentos son

- **text.** Un vector de tipo carácter
- **con**. El nombre de un archivo. Su valor predeterminado es la salida estándar.
- **sep**. Una cadena de caracteres que se escribe tras cada línea de texto. El valor predeterminado es **\n**.

Además de las funciones anteriores que consideran los archivos en modo texto es posible tratarlos en modo binario con las funciones **readBin**, **writeBin**, **readChar** y **writeChar**.

4.1.3. Función unlink

Esta función permite eliminar un archivo o un directorio. Sus argumentos son

- **x**. Un vector de caracteres con los nombres de los archivos y directorios que se desea borrar. Es posible utilizar abreviaturas para especificar conjuntos.
- **recursive**. Valor lógico que indica si los directorios deben borrarse recursivamente o no. El valor predeterminado es FALSE, por lo que no se borran incluso aunque estén vacíos.

Así, si desea borrar el archivo antes creado puede utilizar la orden

```
> unlink("prueba.txt")
```

4.1.4. Gestión de archivos

La gestión de archivos puede realizarse con un grupo de funciones dedicadas:

- **file.create**, crea los archivos que se le indican,
- **file.exists**, indica si existen,
- **file.remove**, elimina los archivos,

- **file.rename**, modifica el nombre de un archivo,
- **file.append**, añade un archivo al final de otro,
- **file.copy**, copia un archivo en otro con la opción de añadirlo,
- **file.symlink**, establece un enlace a un directorio,
- **dir.create**, crea un directorio,
- **file.info**, devuelve información sobre el archivo,
- **file.path**, devuelve el camino absoluto correspondiente al archivo,
- **file.access**, devuelve los permisos disponibles sobre un archivo,
- **file.show**, muestra el contenido de los archivos,
- **list.files**, muestra los nombres de los archivos de un directorio que se corresponden con una abreviatura especificada,
- **path.expand**{, expande un camino especificado relativamente mediante una tilde inicial que indica el directorio inicial del usuario,
- **basename** y **dirname**, devuelven respectivamente la parte de nombre de archivo y nombre de directorio correspondientes al camino que especifica un archivo,
- **file.choose**, muestra un cuadro de diálogo para seleccionar un archivo, y
- **choose.files**, para seleccionar varios.

4.2. Lectura de hojas de datos

Existen diversas funciones que permiten leer datos de un archivo y almacenarlos directamente en una hoja de datos. Son las funciones que más se utilizan para leer datos *bien preparados* y proceder a su análisis. Los datos del archivo deben estar en forma de tabla y se creará una hoja de datos con el mismo número de variables que campos o columnas haya en el archivo y con el mismo número de filas que líneas haya en el archivo.

Estas funciones son **read.table**, **read.csv**, **read.csv2**, **read.delim** y **read.delim2**. La más completa es la primera, siendo el resto modificaciones de la misma simplificadas para leer datos en casos concretos.

La sintaxis es la siguiente:

- **read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".", row.names, col.names, as.is = FALSE, na.strings = "NA", colClasses = NA, nrows = -1, skip = 0, check.names = TRUE, fill = !blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "#")**

Donde los parámetros tienen el siguiente significado:

- **file** es el nombre del archivo a leer, que debe tener una línea por individuo. Si el nombre es relativo, se construye a partir del directorio de trabajo.
- **header** es un valor lógico. Si es TRUE, la primera línea del archivo contiene los nombres de las variables. Si es FALSE, y la primera línea tiene un campo menos que las demás, también se interpreta como nombres de variables. En caso contrario se interpreta como valores del primer individuo. Es conveniente especificar este campo explícitamente.

- **sep** Separador entre campos. El valor predeterminado es uno o varios espacios en blanco.
- **dec** es el separador decimal.
- **quote** es el conjunto de caracteres que delimitan cadenas de caracteres.
- **row.names** permite especificar nombres para las filas. Puede hacerlo de dos modos: Dando un vector de cadenas de caracteres de la misma longitud que el número de filas o mediante un nombre o número que indica un campo del archivo que contiene los nombres.
- **col.names** permite especificar nombres para las columnas o variables. Si no los indica aquí, ni están incluidos en la primera fila del archivo, se construyen con la letra **V** y el número de columna.
- **as.is** vector de valores lógicos que indica cómo tratar los campos no numéricos. El valor predeterminado es **F** con el cual dichos campos se transforman en factores, salvo que se utilicen como nombres de filas. Si este vector es de longitud inferior al número de campos no numéricos se repetirá las veces necesarias. También puede ser un vector numérico que indica que columnas deben conservarse como caracteres.
- **na.strings** es un vector de caracteres y controla con qué valor se indica un valor **NA**. Así, podrá utilizar ***** para leer este tipo de datos procedentes de BMDP.
- **colClasses** es un vector de caracteres que indica la clase de cada columna.
- **nrows** Número máximo de filas que se leerán.
- **skip** indica el número de líneas del principio del archivo que deben saltarse sin leer.
- **check.names**. De tipo lógico, si es TRUE se comprueba que los nombres de variables son correctos y que no hay duplicados.
- **fill**. De tipo lógico, si es TRUE se completan las filas con blancos si es necesario.
- **strip.white**. De tipo lógico, si es TRUE y se ha definido un separador, se eliminan los espacios en blanco al principio y final de los campos de tipo carácter.
- **blank.lines.skip** De tipo lógico, si es TRUE se ignoran las líneas en blanco.
- **comment.char** Indica un carácter a partir del cual no se lee la línea, interpretándose como un comentario.

Aunque esta función es menos versátil que la función **scan** es la que recomendamos que se utilice, preparando los archivos de datos para que puedan ser leídos con ella de modo sencillo.

Por otra parte, la función **write.table** escribe una hoja de datos en un archivo con un formato compatible con su lectura con **read.table**.

Ejemplo:

La siguiente orden lee el archivo **c:/datos/datos2.txt** y produce una hoja de datos, que se almacena en la hoja de datos **h.datos2**

```
> h.datos2 <- read.table("c:/datos/datos2.txt",header=T,as.is=T)
Peso Altura Edad Sexo
Juana Garcia 77 1.63 23 M
Silvia Lopez 58 1.63 23 M
Andres Garces 89 1.85 26 H
```

Laura Perez 55 1.62 23 M
Adela 47 1.60 26 M
Yolanda Lopez 60 1.63 26 M
Lola Martinez 54 1.70 22 M
Alberto Garcia 58 1.65 23 H
Pedro Vera 75 1.78 26 H
Diego Moreno 65 1.70 24 H
Julio Angulo 82 1.77 28 H
Juan Trucha 85 1.83 42 H
Rafael Perez 75 1.74 25 H
Monica Sanchez 65 1.65 26 M

Las siguientes órdenes también leen el archivo **c:/datos/datos2.txt** y producen la hoja de datos **h.datos2**, pero cambiando el directorio de trabajo

```
> opar <- getwd()
> setwd("c:/datos")
> h.datos2 <- read.table("datos2.txt",header=T,as.is=T)
> setwd(opar)
```

4.3. Gestión de objetos

Existen diferentes funciones que permiten almacenar y recuperar objetos, unas son binarias (rápidas y exactas) y otras textuales (legibles por el usuario y compatibles entre plataformas y versiones del lenguaje).

4.3.1.dump

dump produce representaciones en modo texto de un grupo de objetos. Su sintaxis es

dump(list, file = "dumpdata.R", append = FALSE, control = "all", envir parent.frame(), evaluate = TRUE)

donde **list** es un vector de nombres de objetos (de tipo carácter) y **file** es el nombre del archivo en que se guardarán, que si viene dado por un nombre relativo se construirá desde el directorio de trabajo. **append** indica si se añade al contenido del vector o no. La función devuelve el vector de nombres de objetos.

Las representaciones realizadas incluyen el nombre de los objetos y pueden recuperarse con la orden **source** aunque es posible que el modo de los datos numéricos sea modificado y, además, si se recuperan objetos grandes pueden necesitarse grandes cantidades de memoria. En caso de objetos complejos puede que plantee problemas.

Mediante este procedimiento podrá exportar objetos con la finalidad de enviarlos a R en otro ordenador, posiblemente con otro sistema operativo.

4.3.2.write

La función **write** permite escribir un vector o una matriz en un archivo. Solamente escribe los elementos. La matriz necesita ser traspuesta para que coincida con la representación interna de R. Esta orden es cómoda para leer posteriormente los datos con **scan** o desde

otro programa. Su parámetros son:

x. El objeto a escribir (vector o matriz)

file. El nombre del archivo

ncolumns. Número de columnas para escribir

append. Valor lógico que indica si los datos deben añadirse a los existentes en el archivo.

4.3.3. dput y dget

La función **dput** también permite almacenar en un archivo externo, en ASCII, la definición de un objeto. La diferencia fundamental con **dump** es que no conserva el nombre del objeto, sino sólo la definición del mismo. La recuperación se realiza con **dget**. La sintaxis de ambas funciones es

```
dput(x, file = "", control = "showAttributes")
dget(file)
```

4.3.4. save y load

La función **save** almacena los objetos en representación binaria. La recuperación de los datos se realiza con la función **load**. Si desea almacenar todos los objetos, puede utilizar la forma abreviada **save.image**.

La sintaxis es

- **save(..., list = character(0), file = stop("'file' must be specified"), ascii = FALSE, version = NULL, envir = parent.frame(), compress = FALSE)**
- **save.image(file = ".RData", version = NULL, ascii = FALSE, compress = FALSE, safe = TRUE)**
- **load(file, envir = parent.frame())**
- **loadURL(url, envir = parent.frame(), quiet = TRUE, ...)**

donde los argumentos de **save** son

- **...** Los nombres de los objetos que se desea salvar y que se buscaran en el entorno
- **list** Vector de tipo carácter con los nombres de los objetos que se desea salvar y que se buscaran en el entorno
- **file** Nombre del archivo donde se almacenarán los datos
- **ascii** Valor lógico, si es **TRUE** se almacenará en ASCII, aunque no es recomendable.
- **version** La del espacio de trabajo. En la versión actual es 2.
- **envir** Entorno en el que se buscan los objetos que se van a almacenar
- **compress** Valor lógico que indica si los datos deben comprimirse.
- **safe** Valor lógico, si es **TRUE** se salvan los datos en un fichero temporal y si todo va bien, luego se cambia el nombre al mismo. En caso contrario se trabaja directamente con el archivo.

- **name** nombre del archivo
- **quiet** Valor lógico que indica si debe imprimirse algún mensaje (FALSE) o no (TRUE)

La recuperación de los objetos salvados se realiza con la función **load** que, de modo predeterminado, recupera los datos en el entorno actual.

Ejemplos

La siguiente orden guarda todos los objetos de la primera posición de la lista de búsqueda, en el archivo **Total.R** del directorio de trabajo (Posiblemente el archivo resultante sea muy grande).

```
>dump(ls(), "Total.R")
```

Las siguientes órdenes crean el objeto **x**, lo muestran en pantalla , lo salvan, lo eliminan, comprueban que no existe en el entorno de trabajo, y vuelven a recuperarlo.

```
> x <- (1:10)^2
> x
[1] 1 4 9 16 25 36 49 64 81 100
> dump("x")
> rm(x)
> x
Error: Object "x" not found
> source("dumpdata")
> x
[1] 1 4 9 16 25 36 49 64 81 100
```

El objeto ha sido salvado en ASCII en el archivo **dumpdata** con el siguiente contenido:

```
>"x"<- c(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

Las siguientes órdenes realizan las mismas acciones

```
> x <- (1:10)^2
> x
[1] 1 4 9 16 25 36 49 64 81 100
> dput(x,file="x.txt")
> rm(x)
> x
Error: Object "x" not found
> x <- dget("x.txt")
> x
[1] 1 4 9 16 25 36 49 64 81 100
```

El objeto ha sido salvado en ASCII en el archivo **x.txt** con el siguiente contenido:

```
c(1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

Por último, realizamos el mismo conjunto de acciones con **write**:

```
> x <- (1:10)^2
> x
[1] 1 4 9 16 25 36 49 64 81 100
> write(x,file="x.txt")
> rm(x)
> x
Error: Object "x" not found
> x <- scan("x.txt")
> x
[1] 1 4 9 16 25 36 49 64 81 100
```

El objeto ha sido salvado en ASCII en el archivo **x.txt** con el siguiente contenido:

```
1 4 9 16 25
36 49 64 81 100
```

4.4. Creación de archivos temporales

Durante la ejecución de una función es posible necesitar crear un archivo, del que se garantice que no existe en el momento de la ejecución. Para ello R dispone de la función **tempfile** que permite disponer de uno o varios nombres de archivo que no existen en el momento de ejecutar la función y de la función **tempdir** para obtener el directorio temporal que utiliza el sistema operativo.

La sintaxis es la siguiente:

```
tempfile(pattern = "file", tmpdir = tempdir())
```

La función devuelve tantos nombres como longitud tenga el vector **pattern**, nombres que comenzarán respectivamente con la cadena de caracteres que se le suministra en cada componente del vector.

```
> tempfile(c("A","B","PEPE"))
[1] "C:\\DOCUME~1\\Andres\\CONFIG~1\\Temp\\Rtmp460\\A23520"
[2] "C:\\DOCUME~1\\Andres\\CONFIG~1\\Temp\\Rtmp460\\B25402"
[3] "C:\\DOCUME~1\\Andres\\CONFIG~1\\Temp\\Rtmp460\\PEPE24824"
> tempdir()
[1] "C:\\DOCUME~1\\Andres\\CONFIG~1\\Temp\\Rtmp460"
```

Las órdenes siguientes realizan las siguientes acciones: Se calcula un nombre de archivo inexistente y se almacena en la variable **Temp**. Se dirige la salida estándar a dicho archivo, se obtiene una lista de objetos (que se almacena en el archivo citado), y de nuevo se dirige la salida estándar a pantalla. Se borra el archivo, dirigiendo la salida a la variable **SeBorra**, se comprueba que se ha borrado (la variable vale 0) y se eliminan las dos variables usadas.

```
> Temp<- tempfile(pa="Temp")
> sink(Temp)
> ls()
> sink()
```

```
> SeBorra <- unlink(Temp)
> SeBorra
[1] 0
> rm(Temp)
> rm(SeBorra)
```

Ejemplos

A continuación, se crean los objetos **x** y **f**, que se salvan y posteriormente se eliminan. Se recuperan desde el entorno interactivo, y también desde dos funciones, para comprobar las diferencias de almacenamiento en el entorno. En la función **g** se recuperan en el entorno de la función y, al terminar esta, desaparecen. En la función **gg** se recuperan en el entorno global, por lo que al terminar la función siguen existiendo. En este último caso, si existiesen objetos con los mismos nombres en el entorno global, habrían sido sustituidos por los nuevos.

```
> x <- 1:10
> f <- function(x) x^3
> save(x,f,file="Salvado")
> rm(x)
> rm(f)
> load(file="Salvado")
> x
[1] 1 2 3 4 5 6 7 8 9 10
> f
function(x) x^3
> rm(x)
> rm(f)
> g <- function()
+ {
+ load(file="Salvado")
+ return(x,f)
+ }
> g()
$x
[1] 1 2 3 4 5 6 7 8 9 10

$f
function(x) x^3

> x
Error: Object "x" not found
> f
Error: Object "f" not found
> gg <- function()
+ {
+ load(file="Salvado",globalenv())
+ return(x,f)
+ }
> gg()
$x
```

```
[1] 1 2 3 4 5 6 7 8 9 10

$f
function(x) x^3

> x
[1] 1 2 3 4 5 6 7 8 9 10
> f
function(x) x^3
```

4.5. Función library

Esta es una función fundamental. **library** gestiona los libros de la biblioteca, dando información sobre los existentes y cargándolos en memoria o descargándolos de la misma. Un libro está formado por funciones, datos y documentación; todos ellos contenidos en un directorio que contiene varios subdirectorios, con informaciones diversas, de acuerdo con un formato establecido. De hecho, al cargar R, se carga al menos un primer libro denominado **base**.

La sintaxis es

library(package, help = NULL, lib.loc = .lib.loc, character.only = FAL
logical.return = FALSE, warn.conflicts = TRUE)

cuyo significado es:

- **package** es el nombre del libro.
- **help** si se le da el nombre de un libro, devuelve información sobre los componentes del mismo.
- **lib.loc** es una cadena de caracteres que describe la posición de las bibliotecas para buscar en ellas.
- **logical.return** es un valor lógico que devuelve si ha tenido éxito o no la acción.
- **warn.conflicts** es un valor lógico que indica que se impriman los mensajes sobre conflictos al conectar el libro, salvo que este contenga un objeto `.conflicts.OK'.
- **quietly** es un valor lógico que indica que no se dé mensaje si no existe el libro en la biblioteca.
- **libname** es una cadena de caracteres que contiene el nombre del directorio donde se encuentra el libro.
- **pkgname** es una cadena de caracteres que contiene el nombre del libro.
- **all.available** es un valor lógico que indica que se devuelva en un vector los nombres de todos los libros disponibles en **lib.loc**.

Si se llama a la función sin argumentos, devuelve una breve descripción de los diferentes libros disponibles en cada biblioteca de **lib.loc**.

Ejemplos

library() devuelve una lista de los libros que hay en la biblioteca predeterminada de R.

library(help="base") devuelve una pequeña ayuda sobre el libro **base**.

En la ventana que aparece con la información puede observar una lista de los objetos que componen el libro. Puesto que este libro en concreto está en memoria, basta que utilice la ayuda sobre uno de los objetos, por ejemplo **help(Arithmetic)**, para acceder a información detallada sobre el mismo.

Si consulta la lista de búsqueda observará los libros que hay actualmente en memoria

```
> search()
[1] ".GlobalEnv" "package:stats" "package:graphics"
[4] "package:grDevices" "package:utils" "package:datasets"
[7] "package:methods" "Autoloads" "package:base"
```

Si desea añadir el libro **cluster** (que suponemos está en memoria y que ha aparecido al ejecutar la orden **library()** anterior) basta con que escriba la orden **library(cluster)** y podrá observar que se ha incluido en la lista ocupando, como es habitual, la segunda posición:

```
> search()
[1] ".GlobalEnv" "package:cluster" "package:stats"
[4] "package:graphics" "package:grDevices" "package:utils"
[7] "package:datasets" "package:methods" "Autoloads"
[10] "package:base"
```

Para obtener una pequeña ayuda sobr el contenido del libro escriba

```
library(help=cluster)
```

(aunque no esté cargado en memoria, basta con que esté en la biblioteca). El primer objeto que aparece en esta ayuda es **agnes**. Por tanto, una vez cargado el libro en memoria, puede utilizar la orden **help(agnes)** para acceder a la información correspondiente a este libro.

Cuando ya no necesite utilizar el libro **cluster**, descárguelo con la orden **detach()**. Recuerde que esta orden elimina la segunda componente de la lista, así que no la use una segunda vez, ya que eliminaría el siguiente libro (**stats** en este ejemplo).

La instalación de un libro en el sistema consiste en copiar el directorio que lo contiene en un lugar accesible. El método más sencillo consiste en descomprimir el archivo **zip** en que se distribuye en el directorio **library**, dentro de la estructura de R.

Si lo desea puede instalar un libro en cualquier directorio accesible desde el ordenador y luego, cada vez que desee utilizarlo, indicar el camino de dicho directorio, que será entendido como una biblioteca. Así, por ejemplo, si descomprime el archivo **random_0.1.2.zip** en el directorio **C:\lib**, se creará un directorio denominado **random**. Si desea cargar este libro en memoria puede hacerlo con la orden

```
library(random,lib.loc="C:/lib")
```

Del mismo modo, si desea saber qué libros hay en esa biblioteca, puede utilizar la orden

library(lib.loc="C:/lib")

Si observa el directorio **C:\lib\random**, encontrará el archivo de ayuda, **random.chm** y diversa documentación sobre este libro. La estructura de un libro puede encontrarla en el manual *Writing R Extensions* que se suministra con R.

La función **.libPaths** permite ver o modificar la lista de bibliotecas disponibles, que es una vector de cadenas de caracteres. La variable **.Library** contiene el camino a la biblioteca predeterminada del sistema.