Taller de Creación de Paquetes e Informes con R: Aspectos Computacionales

Introducción a la programación y creación de librerías para R

Marta Sestelo[1] y Nora M. Villanueva[2]

[1]sestelo@uvigo.es [2]nmvillanueva@uvigo.es

Departamento de Estatística e I. O.
Universidade de Vigo

Vigo, 15.3.2013

SiDOR
Statistical Inference
Decision & Operations Research Group
Universidade Vigo

Introducción

- Por qué crear un paquete de R?
  - Transferencia y uso de la metodología por parte de la comunidad científica.
  - Documentar el trabajo realizado.
  - Dar ejemplos claros sobre la metodología desarrollada.
  - Organizar el trabajo.
  - Es la forma establecida para contribuir al crecimiento de R.

- Manuales para la creación de paquetes de R:
  - Writting R extensions de R Core Team, Comprehensive R Archive Network (CRAN).
  - Creating R Packages: A Tutorial de Friedrich Leisch.
  - Software for data analysis. Programing with R de John M. Chambers.
- R fue diseñado en un entorno Unix que incluye conjuntos de herramientas (compiladores, utilidades de programación y rutinas).
  - Windows carece de estos componentes. Es necesario instalar las Rtools (emulador de Linux con Cygwin).
  - Mac OS X y Linux no requieren instalar ninguna herramienta adicional (sí en el caso de construir un paquete con código compilado).

- Qué es un paquete de R?
  - Es una colección de código fuente y de otros ficheros (funciones de R y su documentación).
  - Una vez instalado, permite al usuario trabajar con el software por medio de la función `library()`.
  - Durante la creación, los ficheros son organizados de una manera estándar en un directorio fuente (*source*).
  - Para su distribución, el directorio fuente se convierte a un archivo comprimido utilizando la herramienta `build`.
  - El paquete no se utiliza ni desde el directorio fuente ni desde el fichero comprimido.
  - El código fuente es utilizado para generar una versión instalada del paquete en otro directorio.

- R ofrece herramientas básicas para llevar a cabo todo este procedimiento: construir, instalar, comprobar...

- Tipos de paquetes
  - Base packages: parte de la estructura fuente de R, mantenidos por el R Core Team.
  - Recommended packages: parte de cada instalación de R, pero no necesariamente mantenidos por el R Core Team.
  - Contributed packages: el resto. Lo que no significa que estos paquetes sean necesariamente de menor calidad que los anteriores, de hecho muchos de estos paquetes son escritos y mantenidos por miembros del R Core Team.

Ejemplo de paquete

- Ejemplo de regresión lineal simple (similar a la función `lm()`).
  - Función principal `reglineal`
  - Función `print`
  - Función `summary`
  - Función `plot`

```
R> reglin <- function(x, y){
+
+  #calculamos beta   (X'X)^(-1) X'Y
+  beta <- (solve(t(x) %*% x)) %*% t(x) %*% y
+  coef <- beta[, 1]
+
+  #valores ajustados
+  fit <- as.vector(x %*% beta)
+
+  #obtenemos los residuos
+  res <- y - fit
+  n <- nrow(x); p <- ncol(x); df <- n - p
+  var_err <- sum(res**2)/df
+
+  #calculomos matriz covarianzas
+  vcov <- var_err*solve(t(x) %*% x)
+
+  #resultados
+  list(coefficients = coef, fitted.values = fit, residuals = res,
+ var_err = var_err, vcov = vcov, df = df)
+ }
```

```
R> datos <- read.table("production.txt", header = TRUE)

R> head(datos)
 Case RunTime RunSize
1    1     195     175
2    2     215     189
3    3     243     344
4    4     162      88
5    5     185     114
6    6     231     338
```

```
R> reglin(x = cbind(1,datos$RunSize), y = datos$RunTime)
$coefficients
[1] 149.7477030   0.2592431

$fitted.values
[1] 195.1152 198.7447 238.9273 172.5611 ...

$residuals
[1]  -0.1152469  16.2553496   4.0726679 ...

$var_err
[1] 264.1431

$vcov
           [,1]          [,2]
[1,] 69.3581059 -0.278319459
[2,] -0.2783195  0.001379526

$df
[1] 18
```

```
R> lm(datos$RunTime~datos$RunSize)

Call:
lm(formula = datos$RunTime ~ datos$RunSize)

Coefficients:
 (Intercept)  datos$RunSize
    149.7477         0.2592
```

- Los resultados numéricos son iguales pero:
  1. Formato mejor con `lm()`.
  2. Uso de la función `summary()`.
  3. Utilizar fórmulas para especificar el modelo (intercept, interacción, etc).
  4. Poder incorporar covariables categóricas.

- Programación orientada a objetos: ayuda con los puntos 1 y 2.
- Uso de "formulas": ayuda con los puntos 3 y 4.

1. Programación orientada a objetos

- `reglin` devuelve una lista con varios elementos (estimación parámetros, desviación estandar, etc.).
- Para nosotros está claro que es el ajuste de un modelo lineal pero...
- Al ordenador nadie le ha dicho esto, simplemente es una lista con vectores.
- Para solucionarlo se recurre a:
  - Objectos: entidades que R crea y manipula. Se almacenan por nombre y pueden ser de muchos tipos (variables, cadenas, etc.).
  - Clases: definen como los objetos de un cierto tipo son (y son tratados). Ej.: `data.frame`, `matrix`, `lm`, `gam`, etc.
  - Métodos: definen funciones especiales que operan sobre los objetos de una clase particular. Para una función (genérica) existen varios métodos.

- Ejemplo de método:

```
R> x <- rep(0:1, c(10, 20))
R> x
[1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 ...
R> class(x)
[1] "integer"
R> summary(x)
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.0000  0.0000  1.0000  0.6667  1.0000  1.0000

R> y <- as.factor(x)
R> class(y)
[1] "factor"
R> summary(y)
01 10 20
```

- Para una función genérica, por ejemplo `summary()`, existen 2 métodos que realizan distintas operaciones en objetos de distintas clases.

- Como conocer funciones genéricas?

```
R> print
function (x, ...)
UseMethod("print")
<bytecode: 0x107256720>
<environment: namespace:base>

R> summary
function (object, ...)
UseMethod("summary")
<bytecode: 0x100e1b228>
<environment: namespace:base>

R> plot
function (x, y, ...)
UseMethod("plot")
<bytecode: 0x100ee4d40>
<environment: namespace:graphics>
```

- Como conocer los métodos asociados a las funciones genéricas?

```
R> methods(summary)
[1] summary.aov           summary.aovlist       summary.aspell*
[4] summary.connection    summary.data.frame    summary.Date
[7] summary.default       summary.ecdf*         summary.factor
[10] summary.glm           summary.infl          summary.lm
[13] summary.loess*        summary.loglm*        summary.manova
[16] summary.matrix        summary.mlm           ...
```

```
R> methods(sample)
[1] sample.int
Mensajes de aviso perdidos
In methods(sample) : function 'sample' appears not to be generic
```

- Cuando una función genérica `fun` es llamada por un objeto de la clase `pepita`, R busca la función `fun.pepita`.
- Si no se encuentra el método apropiado, R buscará la función `fun.default`.
- Si no se encuentra ningún método por defecto, entonces dará un error.

Importante para programar métodos:

- Un método debe tener todos los argumentos que tiene el genérico incluyendo "..."
- Un método debe tener los argumentos exactamente en el mismo orden que el genérico.
- Sin un genérico tiene argumentos especificados por defecto, todos los métodos deberían usar los mismos valores por defecto.

Estaría bien...

- Si creamos una función que devuelva un objeto de una clase recién creada (p.ej. `reglineal`), deberíamos crear "como mínimo" las funciones `print.reglineal()`, `summary.reglineal()`, `plot.reglineal()` que son las más habituales.

Particularidad de la función `summary()`: los métodos para `summary()` no deberían "printear" (sacar por pantalla) nada. Se utiliza para calcular mas cosas (ver `summary.lm()`). Debemos crear una función `print.summary.reglineal()`.

Ahora definimos clases y métodos para nuestro ejemplo

- Creamos una función genérica `reglineal()`.

```
R> reglineal <- function(x, ...) UseMethod("reglineal")
```

- Añadimos un método por defecto creando una función `reglineal.default()`.

```
R> reglineal.default <- function(x, y, ...){
+
+  x <- as.matrix(x)
+  y <- as.vector(y)
+  res <- reglin(x, y)
+  res$data <- cbind(y, x) #guardamos los datos para el plot
+  res$call <- match.call() #guarda la llamada a la función
+  class(res) <- "reglineal" #definimos la clase
+  return(res)
+ }
```

```
R> reglineal(x = cbind(1,datos$RunSize), y = datos$RunTime)
$coefficients
[1] 149.7477030   0.2592431

$fitted.values
[1] 195.1152 198.7447 238.9273 172.5611 ...

$residuals
[1]  -0.1152469  16.2553496   4.0726679 ...

$var_err
[1] 264.1431

$vcov
           [,1]          [,2]
[1,] 69.3581059 -0.278319459
[2,] -0.2783195  0.001379526

$df
[1] 18
```

```
$data
       y
[1,] 195 1 175
[2,] 215 1 189
...

$call
reglineal.default(x = cbind(1, datos$RunSize), y = datos$RunTime)

attr(,"class")
[1] "reglineal"
```

- Definimos un nuevo método `print.reglineal()` para la función genérica `print()`.

```
R> print.reglineal <- function(x, ...){
+ cat("Call:\n")
+ print(x$call)
+ cat("\nCoefficients:\n")
+ print(x$coefficients)
+ }
```

```
R> reglineal(x = cbind(1,datos$RunSize), y = datos$RunTime)
Call:
reglineal.default(x = cbind(1, datos$RunSize), y = datos$RunTime)

Coefficients:
[1] 149.7477030   0.2592431
```

- Definimos `summary.reglineal()` y el `print.summary.reglineal()`.

```
R> summary.reglineal <- function(x, ...){
+
+  #inferencia coef
+  et <- sqrt(diag(x$vcov))
+  tvalues <- abs(as.vector(x$coefficients))/et
+  pvalues <- 2*pt(tvalues, x$df, lower.tail = FALSE)
+
+  tabla <- cbind(Estimate = coef(x),
+                 StdErr = et,
+                 t.values = tvalues,
+                 p.values = pvalues)
+
+  res <- list(call = x$call, coefficients = tabla)
+
+  class(res) <- "summary.reglineal"
+  return(res)
+ }
```

```
R> model <- reglineal(x = cbind(1, datos$RunSize), y = datos$RunTime)

R> summary(model)
$call
reglineal.default(x = cbind(1, datos$RunSize), y = datos$RunTime)

$coefficients
      Estimate     StdErr  t.values     p.values
[1,] 149.7477030 8.32815141 17.980905 5.996749e-13
[2,]   0.2592431 0.03714198  6.979788 1.614863e-06

attr(,"class")
[1] "summary.reglineal"
```

- Atención a la función `printCoefmat()`.

```
R> print.summary.reglineal <- function(x, ...){
+  cat("Call:\n")
+  print(x$call)
+  cat("\n")
+
+  printCoefmat(x$coefficients, P.value=TRUE, has.Pvalue=T)
+ }

R> summary(model)
Call:
reglineal.default(x = cbind(1, datos$RunSize), y = datos$RunTime)

      Estimate     StdErr t.values  p.values
[1,] 149.747703   8.328151  17.9809 5.997e-13 ***
[2,]   0.259243   0.037142   6.9798 1.615e-06 ***
---
Signif. codes:   0 *** 0.001 ** 0.01 * 0.05 . 0.1  1
```

Nota: separar el cálculo de la salidas de pantalla tiene ventajas ya que podremos utilizar los valores del calculados en el `summary` para cálculos posteriores.

```
coef(summary(model))[,4]
[1] 5.996749e-13 1.614863e-06
```

2. Uso de "formula"

- Formulación en nuestro ejemplo:

  `reglineal<-function(formula,data, ...)`

- Hace que la función creada sea fácil de usar ya que tiene la sintaxis igual a otras funciones que existen por defecto en R (lm, glm, etc.)

- Evita tener que escribir el vector o matriz de datos, utiliza el argumento `data`.

- Útil para analizar datos completos ya que no es necesarios cogerlos uno a uno dentro de la función.

- Como funciona `formula`?
  - El objeto central que se crea primero a través de la formula es `model.frame`, data frame que contiene únicamente las variables que aparecen en la fórmula.
  - El data frame obtenido contiene un atributo que se llama `terms` que indica si existe una variable dependiente, un intercepto, etc.
  - El `model.frame` y el atributo son a continuación utilizados para construir la matriz de diseño para el modelo con la función `model.matrix()`.
  - La variable respuesta se obtiene con la función `model.response()` y siempre es la primera columna del `model.frame`.
- En la mayoría de los casos funciona con este código:

```
cl <- match.call()
mf <- model.frame(formula = formula, data = data)
mt <- attr(mf, "terms")
x <- model.matrix(mt, data = mf)
y <- model.response(mf)
```

*Nota: `match.call()` se usa para guardar una llamada y usarla después.

- Creamos un nuevo método de la función `reglineal()`.

```
R> reglineal.formula <- function(formula, data = list(), ...){
+  mf <- model.frame(formula = formula, data = data)
+  mt <- attr(mf, "terms")
+  x <- model.matrix(mt, data = mf)
+  y <- model.response(mf)
+
+  res <- reglineal.default(x, y, ...)
+  res$call <- match.call()
+  res$formula <- formula
+  return(res)
+ }
```

```
R> reglineal(RunTime ~ RunSize, data = datos)
Call:
reglineal.formula(formula = RunTime ~ RunSize, data = datos)

Coefficients:
(Intercept)     RunSize
149.7477030   0.2592431
```

```
R> reglineal(RunTime ~ RunSize-1, data = datos)
Call:
reglineal.formula(formula = RunTime ~ RunSize - 1, data = datos)

Coefficients:
 RunSize
0.8601491
```

```
R> data(iris)

R> reglineal(Petal.Length ~ ., data = iris)
Call:
reglineal.formula(formula = Petal.Length ~ ., data = iris)

Coefficients:
     (Intercept)      Sepal.Length     Sepal.Width     Petal.Width
      -1.1109888         0.6080058      -0.1805236       0.6022215
Speciesversicolor  Speciesvirginica
       1.4633709         1.9742229


R> reglineal(Petal.Length ~ Species, data = iris)
Call:
reglineal.formula(formula = Petal.Length ~ Species, data = iris)

Coefficients:
     (Intercept) Speciesversicolor  Speciesvirginica
           1.462             2.798             4.090
```

- Creamos un nuevo método de la función `plot.reglineal()`.

```
R> plot.reglineal <- function(x, ...){
+  data <- x$data
+  coef <- coef(x)
+  lab <- colnames(data)
+  plot(data[, 3], data[, 1], xlab = lab[3], ylab = lab[1], ...)
+  abline(coef[1], coef[2], lwd = 2)
+ }

R> plot(model)
```

*Uso de "..." Se utiliza para no poner todos los argumentos de otras funciones pero que podemos cambiarlos si fuese necesario. Por ejemplo:

```
R> plot(model, col = "red")
```

Control de argumentos

- Sin control de argumentos.

```
R> reglineal(RunTime~RunSize)
Error en eval(expr, envir, enclos) : objeto 'RunTime' no encontrado
R> reglineal.formula <- function(formula, data = list(), ...){
+  if(missing(data)){stop("data argument is requiered")}
+  mf <- model.frame(formula = formula, data = data)
+  mt <- attr(mf, "terms")
+  x <- model.matrix(mt, data = mf)
+  y <- model.response(mf)
+  res <- reglineal.default(x, y, ...)
+  res$call <- match.call()
+  res$formula <- formula
+  return(res)}
```

- Con control de argumentos.

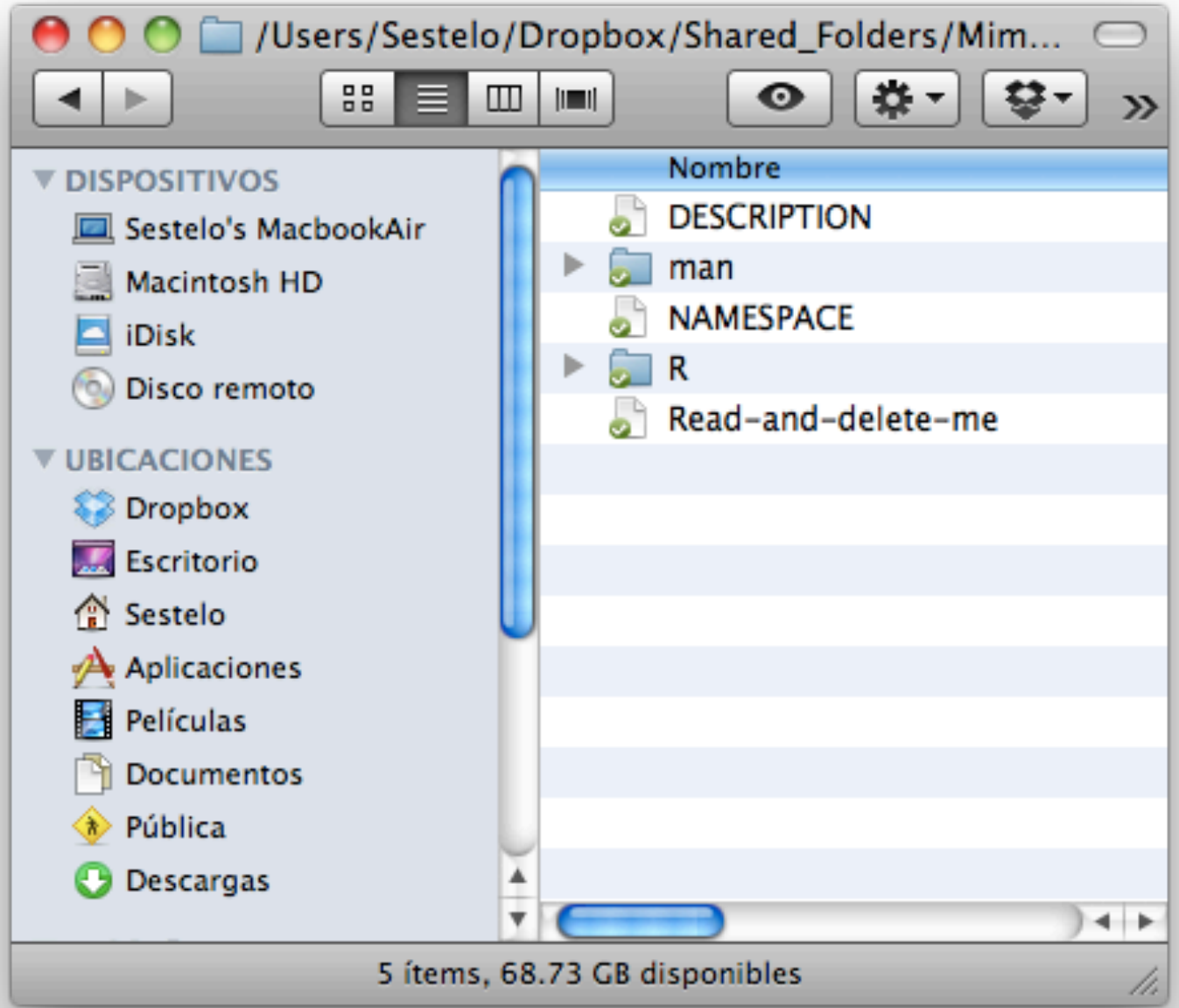
```
R> reglineal(RunTime~RunSize)
Error en reglineal.formula(RunTime ~ RunSize) :
  data argument is requiered
```

Construcción del paquete

- Una vez tenemos todo el código preparado, la creación del paquete se lleva a cabo utilizando la función `package.skeleton()`
- Esta función crea un directorio en el disco duro con el mismo nombre que el paquete y con el siguiente contenido:
  - Un fichero `DESCRIPTION`: descripción del paquete, autor, licencia, etc.
  - Un fichero `NAMESPACE`: carga los paquetes necesarios y exporta las funciones.
  - Un subdirectorio `man/`: documentación de las funciones.
  - Un subdirectorio `R/`: funciones con código R.
  - Un subdirectorio `*data/`: conjuntos de datos.
  - Un subdirectorio `*src`: para código Fortran, C o C++.
- Todos son opcionales, excepto `DESCRIPTION`. Sin embargo, cualquier paquete útil debería incluir al menos `man/`, `R/` y `data/`.

```
R> mylist <- c("reglin", "reglineal", "reglineal.default",
"print.reglineal", "summary.reglineal", "print.summary.reglineal",
"reglineal.formula", "plot.reglineal")


R> package.skeleton(name = "ourpackage", mylist)
Creating directories ...
Creating DESCRIPTION ...
Creating NAMESPACE ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './ourpackage/Read-and-delete-me'.
```


/Users/Sestelo/Dropbox/Shared_Folders/Mim...
DISPOSITIVOS
Sestelo's MacbookAir
Macintosh HD
iDisk
Disco remoto
UBICACIONES
Dropbox
Escritorio
Sestelo
Aplicaciones
Películas
Documentos
Pública
Descargas
Nombre
DESCRIPTION
man
NAMESPACE
R
Read-and-delete-me
5 ítems, 68.73 GB disponibles

- Instrucciones del fichero `Read-and-delete-me`:

```
* Edit the help file skeletons in ‘man’, possibly combining
help files for multiple functions.
* Edit the exports in ‘NAMESPACE’, and add necessary imports.
* Put any C/C++/Fortran code in ‘src’.
* If you have compiled code, add a useDynLib() directive
to ‘NAMESPACE’.
* Run R CMD build to build the package tarball.
* Run R CMD check to check the package tarball.

Read "Writing R Extensions" for more information.
```

Intalación y verificación del paquete

`R CMD command packagename`

INSTALL instala el paquete en una librería y lo hace disponible para su uso en R.

check ejecuta una batería de tests en el paquete. Para poder subir el paquete al CRAN es indispensable que no haya ni un sólo warning.

build elimina los ficheros temporales de la estructura fuente del paquete y lo empaqueta o comprime en un archivo fácil de compartir (entre sistemas!) e instalar sin descomprimir. Hace paquetes fuente pero comprimidos.

Para usuarios Mac OS y Linux: para distribuir un paquete que contiene código Fortran o C con usuarios de Windows, es necesario construir un binary package compilado bajo Windows, que dará como resultado un .zip.

```
R CMD INSTALL --build packagename
```

```
R CMD command packagename
```

- Todos estos comandos se ejecutan desde la Consola de Windows (Inicio/Todos los programas/Accesorios/Símbolo del sistema) o el Terminal (Finder/Aplicaciones/Utilidades). Es necesario estar en el directorio que contiene al paquete fuente.

- Para poder instalar el paquete (en nuestro ordenador) es imprescindible editar las ayudas:
  - Editar el archivo ourpackage-package.Rd
  - Rellenar el argumento `\title{ }`

Documentación

- Muy Importante. Documentación confusa, pobre o inexistente, el paquete no se podrá utilizar.
- Subcarpeta `man`: archivos de documentación "Rd". Para cada función de la carpeta `R`, debemos crear un archivo de documentación.
  - Ej. `mifuncion.R` $\Longrightarrow$ carpeta `R` `mifuncion.Rd` $\Longrightarrow$ carpeta `man`
- R utiliza un lenguaje especial para crear estos archivos de documentación.
- Este lenguaje es parecido al LaTeX .
- Durante la instalación del paquete se traslada este lenguaje genérico "Rd" a archivos HTML y texto.
- Lenguaje consiste en una serie de instrucciones con sus correspondientes argumentos.

```
\command{arg}
```

Documentación de las funciones

- El archivo "Rd" para una función contiene puntos obligatorios y opcionales

`\name{name}`	Nombre al archivo de la ayuda.
`\alias{topic}`	A menudo, múltiples entradas: uno por cada palabra que, cuando se utiliza después de ? conduce a esta página de ayuda.Suele incluir el nombre de la función.
`\title{Title}`	Breve descripción del topic. No caracteres especiales. Primera letra en mayúscula. No punto final.
`\description{...}`	Pocas líneas describiendo el topic.
`\usage{fun(arg1,...)}`	Función exacta sintaxis de llamada, mostrando args.
`\arguments{...}`	Descripción de cada argumento.
`\details{...}`	Descripción detallada de lo que hace la función.
`\value{...}`	Descripción de los valores que devuelve.
`\references{...}`	Bibliografía. Usar `\url{}` para la dirección de la web.
`\author{...}`	Autor. Usar `\email{}` para la dirección de mail.
`\note{...}`	Cualquier anotación que se desee especificar.
`\seealso{...}`	Señalar los objetos de R relacionados, `\code{url{}}`
`\examples{...}`	Ejemplos de como funciona la función.
`\keyword{key}`	Múltiples entradas.

`keyword{}` debe contener como mínimo una de las palabras clave estándar:

1. archivo `R_HOME/doc/KEYWORDS.db`.
   * Windows: `R_HOME` = `c:/Archivos de programa/R/R-2.15.3`
   * Mac: `/Library/Frameworks./R.framework/Versions/2.15/Resources/`
2. R utilizar `file.show(file.path(R.home("doc"), "KEYWORDS"))`

```
GROUPED Keywords
----------------

Graphics
aplot & Add to Existing Plot / internal plot
dplot & Computations Related to Plotting
hplot & High-Level Plots
iplot & Interacting with Plots
color & Color, Palettes etc
dynamic & Dynamic Graphics
device & Graphical Devices
...
```

Documentación del conjunto de datos

- El archivo "Rd" para un conjunto de datos contiene puntos obligatorios y opcionales

`\name{name}`	Nombre del conjunto de datos.
`\docType{data}`	Siempre `data`.
`\alias{topic}`	Igual que para las funciones.
`\title{Title}`	Breve descripción del objeto datos.
`\description{...}`	Pocas líneas describiendo el objeto.
`\usage{name}`	Siempre `name`.
`\format{...}`	Descripción del formato de los datos.
`\details{...}`	Descripción detallada de lo que hace la función.
`\source{...}`	Descripción de la fuente original de los datos.
`\references{...}`	Bibliografía.
`\examples{...}`	Ejemplos de como utilizar el objeto datos.
`\keyword{datasets}`	Siempre `datasets`.

- Para añadir un conjunto de datos

```
R> save(datos, file="/Users/Nora/Desktop/ourpackage/data/datos.Rda")
```

- Para añadir un archivo .Rd

```
R> prompt(datos, file="/Users/Nora/Desktop/ourpackage/man/datos.Rd")
```

Documentación del paquete

- El archivo "Rd" para el paquete contiene puntos obligatorios y opcionales
- Ideas para documentar el paquete: el esqueleto contiene un archivo `ourpackage-package.Rd` con una entrada `alias` para el nombre del paquete y otro similares al `DESCRIPTION`.

`\name{name}`	Nombre del paquete.
`\alias{topic}`	Igual que para las funciones.
`\docType{package}`	Siempre `package`.
`\title{Title}`	Breve descripción del paquete.
`\description{...}`	Pocas líneas describiendo el paquete.
`\author{...}`	Autor(es) y el encargado del mismo.
`\references{...}`	Bibliografía.
`\keyword{package}`	Siempre `package`.
`\examples{...}`	Ejemplos sencillos de las funciones más importantes.

- Extenso conjunto de instrucciones para dar formato. Descritas en el capítulo 2 Writing R Extension.
- La instrucción \arguments usa la subinstrucción \item para organizar una llista.
- Si la instrucción \value es un lista, también utiliza \item.
  Ej. Función reglineal $\Longrightarrow$ deberemos insertar 2 item:

```
\value{
 \item{formula}{formula}
 \item{data}{conjunto de datos}
}
```

- NO debe haber ESPACIOS entre la llave que cierra el nombre del item y la llave que abre el texto del argumento.
  - \item{data}{conjunto de datos}
- En las secciones \references y \source:
  - Separar los párrafos por una línea en blanco para cada referencia.
  - Escribir los nombres de los autores de la forma, Author, A. B. y separar por una como o por and pero nunca ambos. A continuación escribir el año. Título del libro y la revista en \emph{...}, pero no el título del artículo. El volumen de la revista en \bold{...} y una coma. Utilizar el rango de páginas con --.

- Algunas de las más utilizadas son:
  - Utilizar la fuente typewritter: `\code{mifuncion}`
  - Insertar URL: `\url{http://www.r-project.org/}`
  - Insertar una dirección de correo: `\email{pepita@uvigo.es}`
  - Cuando hay funciones relacionadas es útil insertar enlaces entre los diversos archivos de documentación `\code{\link{mifuncion}}`
  - Escribir fórmulas con la instrucción `\eqn{}` o `\deqn{}`.

- Los archivos "Rd" se pueden convertir a otros formatos con los siguientes comandos:
  - `R CMD Rd2txt mifuncion.Rd`
  - `R CMD Rdconv -t=html -o=mifun.html mifuncion.Rd`

    Este comando devuelve la versión en HTML y se puede comprobar que todas las secciones estén bien definidas.

- Trucos para rellenar los archivos "Rd":
  - % es el símbolo del comentario (como en LaTeX).
  - Multiplicar matrices utilizar \ para insertar %, ej. C=A\ %*\ %B.
  - Evitar los caracteres $, #, _, <, > y |.
  - Evitar fórmulas, ya que los documentos presentados con las instrucciones ? o `help` en R son versiones en texto plano. O crear un enlace a un archivo PDF.
  - La ayuda de los nuevos métodos creados, p. ej. `summary.reglineal`, debe ser editada según:

    ```
    \method{generic}{class}{arguments}
    ```

    ```
    \method{summary}{reglineal}{x, ...}
    ```

Edición de DESCRIPTION

El primero y el fundamental es DESCRIPTION: indica las características básicas del paquete y un breve resumen de su funcionalidad.

- Los **campos obligatorios** son los siguientes

```
Package: name
Title: What the package does (short line)
Version: 1.0
Author: Who wrote it
Maintainer: Who to complain to <yourfault@somewhere.net>
Description: More about what it does (maybe more than one line)
     bla, bla, bla, bla, bla...
License: What license is it under?
```

- Editar el archivo con el texto todo seguido. Ej. `Description` ocupa más de una línea, la segunda y siguientes deben empezar con un espacio en blanco o un tabulador.
- `Version` es una secuencia de al menos dos números enteros no negativos separados por un único punto o guión.
- `License` contiene una descripción o abreviaturas como `GPL`, `LGPL`, `BSD`, or `Artistic`.
- Existen **campos opcionales** como `Date`, `Depends`, `URL`, `LazyLoad`, etc.
- `Date` según el estilo `yyyy-mm-dd`.
- `Depends` contiene una lista de nombres de los paquetes de los que depende el que estamos construyendo. Además, se debe indicar si el paquete depende de una cierta versión de R, p. ej. `R(>=2.15.3)`.

Edición de NAMESPACE

Otro archivo importante es NAMESPACE. Se utiliza para manejar los nombres de las funciones entre paquetes.

- Especificar qué funciones de nuestro paquete pueden exportarse para que otro usuario las pueda utilizar en su paquete.

  Ej. `export(fun1,fun2)`

  Ej. `exportPattern("^[[:alpha:]]+")`, por defecto para exportar todas

- Especificar que funciones deben ser importadas de otros paquetes para ser utilizadas en el nuestro. Ej. `importFrom("mgcv",gam)`

- Utilizar `S3method(print, reglineal)` para registrar los métodos S3 que creamos al construir el paquete .

Otros contenidos

- Algunos autores a veces incluyen guías adicionales del paquete, preferiblemente como archivos PDF. Si estas guías están escritas utilizando Sweave se conocen con el nombre de Vignettes (ver `help("vignette")`). Debes crear una carpeta `ins/doc` en el paquete fuente para incluir estos documentos.

- `CITATION` es un fichero donde se incluye la referencia del paquete. Se puede acceder al contenido del fichero desde R utilizando por ejemplo `citation('ourpackage')`.

Envío al CRAN

- Crea un fichero .tar.gz mediante R CMD build

```
R CMD build ourpackage
```

- Comprueba que el paquete puede ser instalado, ejecuta los ejemplos y comprueba que la documentación esté completa mediante `R CMD check --as-cran` sobre el .tar.gz. Asegúrate de que se puede ejecutar todo el proceso sin warnings.

```
R CMD check --as-cran ourpackage.tar.gz
```

- Sube el archivo .tar.gz utilizando "anonymous" como log-in y tu dirección de email como password a

  ftp://cran.R-project.org/incoming

  y envía un mensaje a cran@r-project.org sobre el paquete subido. Los "mantainers"del CRAN (principalmente Kurt Hornik) ejecutarán estos tests antes de ponerlo en el repositorio principal.