R como instrumento de trabajoLa sintaxis de R en la práctica

versión: *Fri Feb 3 14:50:27 2017*

1. Sintaxis de R: breve repaso

En este documento se recogen los aspectos mas relevante del uso de R y que permiten iniciarse en el uso de esta potente herramienta.

2. Expresiones

- R *evalúa* expresiones.
- Las expresiones se escriben siguiendo una sintaxis precisa.
- Existe muchas alternativas para obtener un mismo resultado.
- Las expresiones contienen:
  - valores
  - funciones
  - objetos
  - operadores
  - espacios en blanco: tanto como desee para mejorar la lectura
  - El carácter # genera un comentario a su derecha
- En la expresiones cualquier valor puede ser sustituido por una expresión.

Para definir un procedimiento de cálculo o programa recurrimos a un conjunto de expresiones de acuerdo a un algoritmo. Estos procedimiento son secuenciales, se evalúan las expresiones empezando por la primera, en un proceso secuencial.

3. Valores

Podemos utilizar valores numéricos, lógicos o de texto; operadores aritméticos o lógicos y funciones para realizar un sin fin de procedimientos.

3.1. Valores numéricos

Los valores numéricos en R pueden ser enteros, reales o complejos; pueden aparecer de forma individual o en conjuntos (ver vectores, matrices y otros).

Las operaciones que devuelven una indeterminación proporcionan el valor correspondiente: $-\infty$, se representa por `-Inf`, o ∞, se representa por `Inf`; las que no tiene un resultado numérico devuelven `NaN`: *not is a number*; y cuando el valor no existe encontramos `NA`, *not available* o *missing value*, para indicarlo.

Los siguientes ejemplos muestran el uso de valores numéricos *entreros* y *reales*.

```
> 2
```

```
[1] 2
```

```
> 0.021
```

```
[1] 0.021
```

```
> 1e6       # valor de tipo real
```

```
[1] 1e+06
```

```
> 1e6 + 1   # resultado entero
```

```
[1] 1000001
```

3.1.1. Conversión

```
> 1e7
```

```
[1] 1e+07
```

```
> as.integer( 1e7 )  # Entero muy grande
```

```
[1] 10000000
```

```
> as.integer( 1e10 ) # No cabe en la definición de entero
```

```
Warning: NAs introduced by coercion to integer range
[1] NA
```

3.1.2. Redondeando

```
> 1/3
```

```
[1] 0.3333333
```

```
> options( digits = 3 )
> 1/3
```

```
[1] 0.333
```

```
> log( 2 )  # logaritmo natural base e
```

```
[1] 0.693
```

```
> 1e6 + 1   # resultado entero?
```

```
[1] 1e+06
```

3.1.3. Series

```
> 1:10
```

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

```
> seq( 0, pi, 0.1 )
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
[20] 1.9 2.0 2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8 2.9 3.0 3.1
```

3.1.4. Valores no numéricos

```
> sqrt( -9 )
```

```
Warning in sqrt(-9): Se han producido NaNs
```

```
[1] NaN
```

```
> sqrt( as.complex( -9 ) ) # La raíz de un valor negativo
```

```
[1] 0+3i
```

```
> 1 / 0
```

```
[1] Inf
```

```
> log( -1 )  # No existe el logaritmo
```

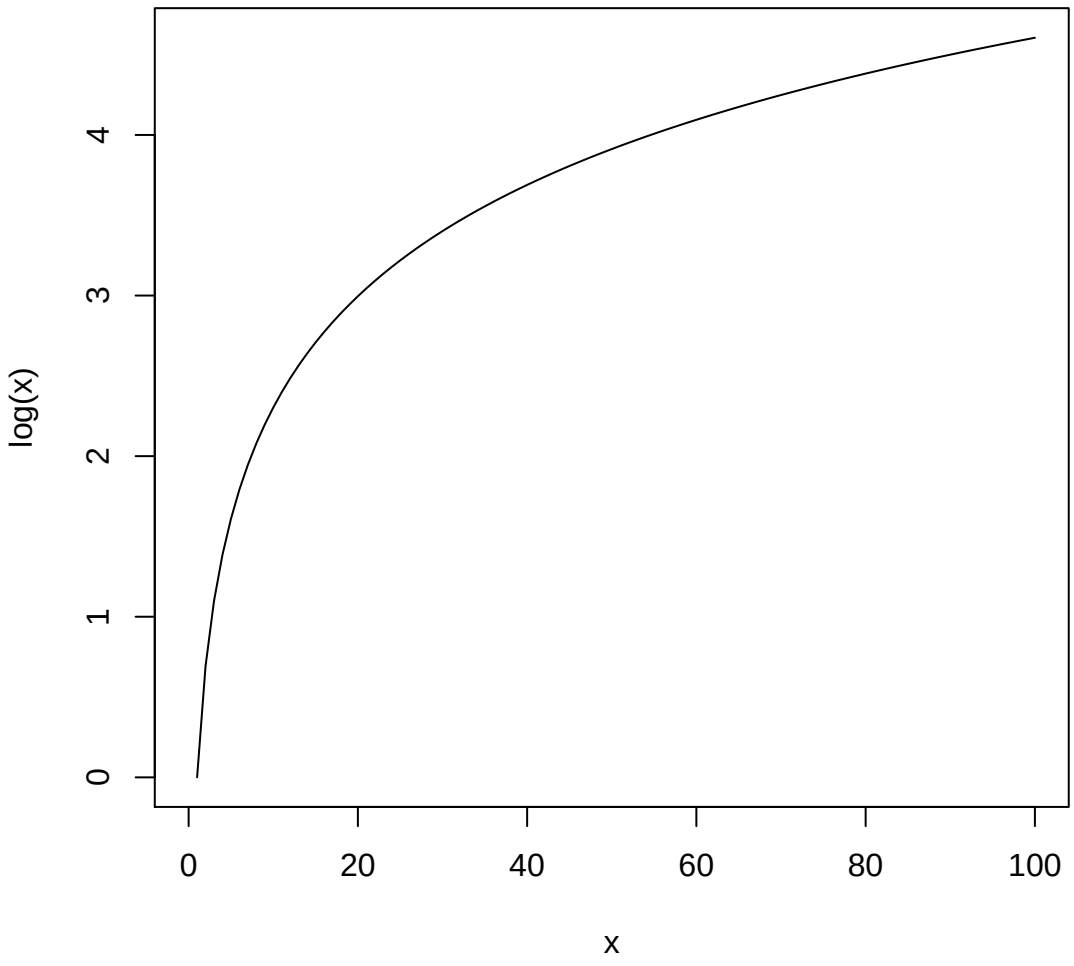
```
Warning in log(-1): Se han producido NaNs
```

```
[1] NaN
```

```
> log( 0 )
```

```
[1] -Inf
```

```
> curve( log, 0, 100 ) # representación de la función logaritmo
```



3.2. Textuales (*string*)

Los valores de texto facilitan manejar la información cualitativa o la producción de mensajes para una lectura humanizada.

```
> "Casa"
```

```
[1] "Casa"
```

```
> "Casa" == "casa"          # Sensibilidad a mayúsculas
```

```
[1] FALSE
```

```
> paste ( "Casa", 1:10, sep = " número " )
```

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

```
> # 1 + "1"                 # ¡No es posible!
>
> # read.table( "http://www.um.es/docencia/emc/datos/biom2003.dat" )
```

4. Operadores

4.1. Operadores aritméticos

En R encontramos los operadores habituales: suma, restas, productos, cocientes y potencias (`+`, `-`, `*`, `/`, `^`) a los que se añaden otros menos conocidos como `:`, para generar series, o `%%` para conseguir el resto del cociente entero, u operadores específicos para producto de matrices (`%*%`). La jerarquía de la operación es también la habitual y puede alterarse con el uso de paréntesis. Ante la duda es conveniente el uso de paréntesis.

```
> 3 ^ 2             # Operador potencia
```

```
[1] 9
```

```
> 10:1              # Operador de serie
```

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

```
> 3 + 2 * 5         # El producto tiene prioridad
```

```
[1] 13
```

```
> 1 / ( 2 * 5 )     # Cociente con numerador "forzado por paréntesis"
```

```
[1] 0.1
```

```
> 1/ 2 / 5          # El mismo cociente anterior
```

```
[1] 0.1
```

4.2. Jerarquía de operación

```
> 3 ^ 2             # Operador potencia
```

```
[1] 9
```

```
> 3 ^ 1 / 2        # La potencia tiene prioridad
```

```
[1] 1.5
```

```
> 3 + 2 * 5        # El producto tiene prioridad
```

```
[1] 13
```

```
> 1 / ( 2 * 5 )    # Cociente con numerador "forzado por paréntesis"
```

```
[1] 0.1
```

```
> 1 / 2 / 5        # El mismo cociente anterior
```

```
[1] 0.1
```

4.3. Operadores y series

```
> 2 ^ 0:8          # La potencia tiene prioridad
```

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

```
> 3 + 2 * 1:5      # La serie tiene prioridad
```

```
[1]  5  7  9 11 13
```

```
> 3 + ( 2 * 1 ):5  # Alteramos la jerarquía
```

```
[1] 5 6 7 8
```

```
> 1:10 %% 2
```

```
 [1] 1 0 1 0 1 0 1 0 1 0
```

4.4. Operadores lógicos

La posibilidad de automatizar procedimientos requiere la evaluación lógica, para ellas se utilizan los operadores lógicos, comparan entre dos expresiones

- `==` `!=`: Igual a, distinto a
- `>` `>=`: mayor, mayor o igual
- `<` `<=`: menor, menor o igual
- `&`: `AND` verdadero si ambas expresiones son ciertas
- `|`: `OR` verdadero si al menos una de las expresiones es cierta
- `!`: `NOY` operador de negación que afecta a una sola expresión

4.5. Igual o distinto

```
> 3 == 2       # Comparación igualdad
```

```
[1] FALSE
```

```
> 5 != 4       # Comparación desigualdad
```

```
[1] TRUE
```

```
> 3 > 2       # Es mayor
```

```
[1] TRUE
```

```
> ! 3 > 2     # Negación
```

```
[1] FALSE
```

```
> ! TRUE
```

```
[1] FALSE
```

4.5.1. Operaciones con valores lógicos

```
> TRUE
```

```
[1] TRUE
```

```
> TRUE  + 1
```

```
[1] 2
```

```
> FALSE + 1
```

```
[1] 1
```

```
> # ¿qué número es verdadero?
> as.logical( 1 )
```

```
[1] TRUE
```

```
> as.logical( 0 )
```

```
[1] FALSE
```

4.5.2. ¿Iguales?

```
> 0 != 0.0000000000000001 # ¿son iguales?
```

```
[1] TRUE
```

```
> 5 %% 2 != 0             # ¿El resto de 5 entre dos es 0?
```

```
[1] TRUE
```

```
> 3 > 2 & 5 %% 2 != 0     # Resultado OR de dos expresiones
```

```
[1] TRUE
```

```
> 1:5 > 2                 # evaluación de un conjunto
```

```
[1] FALSE FALSE  TRUE  TRUE  TRUE
```

5. Funciones

Las funciones podemos considerarlas como programas dentro del programa R; esencialmente están asociadas a un nombre, que sigue las mismas reglas que cualquier objeto de R, y a un conjunto de argumentos, que son propios de cada función.

```
> # nomreDeLaFuncion( argumento1, argumento1, ... )
> # argumento: expresión
> # plot( hclust( dist( swiss ), method = "mcquitty" ), hang = -1 )
```

El tabulador nos es útil tanto para completar el nombre de la función como dentro del paréntesis ya que actúa como ayuda informando de los argumentos esperados.

5.1. *Una función para conocerlas a todas*

```
> help( swiss )
```

5.2. En la práctica

En muchas ocasiones no es necesario indicar más que alguno de los argumentos, lo que facilita el uso.

```
> log( 2 )        # logaritmo natural de 2
```

```
[1] 0.693
```

```
> log( 2, 10 )    # logaritmo en base 10
```

```
[1] 0.301
```

```
> exp( 1 )        # función exponencial
```

```
[1] 2.72
```

```
> log( exp( 1 ) ) # ¡¡¡Uno!!!
```

```
[1] 1
```

5.3. Nombres y orden de los argumentos

```
> log( x = 2, base = 10 ) # indicando el nombre de los argumentos
```

```
[1] 0.301
```

```
> log( base = 10, x = 2 ) # alterando el orden de los argumentos
```

```
[1] 0.301
```

5.4. Más sobre series

```
> seq( 0, 2, 0.1 )
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
[20] 1.9 2.0
```

```
> seq( from = 0, to = 2, by = 0.1 )
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
[20] 1.9 2.0
```

```
> seq( to = 2, by = 0.1, from = 0 )
```

```
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8
[20] 1.9 2.0
```

5.5. Pero ¿cuántas funciones hay?

El número de funciones disponibles en R crece día a día, para buscar información:

- Buscar funciones
- Una estimación actualizada
- ¿Pero cuantas necesito conocer?
  - No solo no necesitamos aprenderlas todas para usar R: ¡es imposible!
  - Una "chuleta" de funciones: R reference card V2
  - Documentación y el buscador no ayudarán mucho para localizar la funciones, y los procedimientos, necesarios

6. Objetos en R

Las ventajas del uso de un lenguaje es la generalización de un procedimiento de cálculo y esto conlleva el uso de variables que contienen los valores particulares de cada caso. Para utilizar una variable es necesario que se asigne, anteriormente, un valor a ella, de lo contrario obtendremos un error como resultado que indica la inexistencia de tal variable.

La asignación ser realiza mediante un operador específico y no debe leerse como igualdad:

- `<-`: "asignar a la izquierda (dos caracteres)
- `->`, "asignar a la derecha
- `=`, "asignar a la izquierda

```
> x <- 3      # creamos el objeto x
>             # que toma el valor de la expresión
> x           # para conocer el valor de un objeto
```

```
[1] 3
```

```
> 1 / 5 -> y
> pi          # valor predefinido
```

```
[1] 3.14
```

6.1. Alternativas de asignación

```
> y <- 5 * 9 / 2          # y toma el valor de la expresión de la derecha
> "Carmolí" -> localidad  # Asignación a la derecha
> n = 30                  # asignación no recomendada
>
> ( m <- 25 ^ 2 )         # evita tener que evaluar la expresión m
```

```
[1] 625
> # para ver su contenido
```

6.2. La asignación es destructiva

```
> i <- 1          # La última asignación el la que vale
> i
```

```
[1] 1
```

```
> i <- i + 1      # Aumentamos el valor de i
>
> ( i <- i + 1 ) # Aumentamos nuevamente el valor de i
```

```
[1] 3
```

6.3. Aprovechando los objetos

```
> control <- TRUE                     # Guardando un valor lógico
> solucion <- sqrt( as.complex( x ) ) # Previniendo valores negativos
>
> pesoIndividuo     <- 70
> longitudIndividuo <- 180            # buenos nombres de objetos
>
> indiceCorporal <- pesoIndividuo / longitudIndividuo
```

6.4. Aprovechando más los objetos

```
> pesoIndividuo     <- c( 110, 100, 125 ) # g:  gramos
> longitudIndividuo <- c( 100, 105, 110 ) # mm: milímetros
>
> indiceCorporal <- pesoIndividuo /
+                   longitudIndividuo # lectura mejorada de expresiones
```

Los nombres de los objetos son un conjunto de caracteres, el primero una letra; las letras permitidas son las del alfabeto inglés; pueden utilizarse las cifras de 0 a 9, y opcionalmente el punto (`.`) o la barra baja (`_`), si bien no son muy recomendables. Debe evitarse que el nombre pueda ser interpretado como una expresión, ya que R devolverá un error.

6.5. Vectores

La potencia de los lenguajes es el uso de conjuntos de valores ordenados, a esto los llamamos vectores; en ellos podemos guardar cualquier tipo de valor pero todos deber ser del mismo tipo.

Habitualmente, los conjuntos de datos grandes se almacenan en fichero de datos, pero, para casos con un número pequeño podemos construir los vectores con ayuda de la función `c()`, que como veremos se utiliza con mucha frecuencia.

Para utilizar todos los elementos de un vector se recurre al nombre del mismo, para trabajar con un elemento en particular o un conjunto de ellos recurrimos a uso de subíndices, de tal modo que `v[1]` es v_1; veamos el siguiente ejemplo:

6.5.1. Definición de un vector y selección de elementos

```
> v <- c( 3, 2, 7, 9 ) # Definimos el vector
>
> v[ 1 ]               # El primer elemento
```

```
[1] 3
```

```
> v[ 1:3 ]             # Los tres primeros elementos
```

```
[1] 3 2 7
```

6.5.2. Más sobre selección

```
> v <- c( 3, 2, 7, 9 )
> i <- c( 1, 3 )
>
> v[ i ]               # Los elementos 1 y 3
```

```
[1] 3 7
```

```
> v[ c( 1, 3 ) ]       # Igual al anterior
```

```
[1] 3 7
```

6.5.3. Selección inversa

```
> v <- c( 3, 2, 7, 9 )
> i <- c( 1, 3 )
>
> v[ -1 ]              # Todos menos el primer elemento
```

```
[1] 2 7 9
```

```
> v[ -i ]
```

```
[1] 2 9
```

6.5.4. Aprovechando los vectores

Las ventajas de tener datos en vectores se derivan de calcular directamente con el vector o aplicarle funciones (a todo a a subconjuntos):

```
> v <- c( 3, 2, 7, 9 )
>
> v * 5          # Producto de todos los elementos
```

```
[1] 15 10 35 45
```

```
> length( v )    # Número de elementos del vector
```

```
[1] 4
```

```
> sum( v )       # Suma de todos los elementos del vector
```

```
[1] 21
```

```
> sum( v ) / length( v ) # ¿quién soy?
```

```
[1] 5.25
```

6.5.5. Encadenando vectores

```
> v <- c( 3, 2, 7, 9 )
> n <- length( v )
>
> # uniendo vectores
> c( v, 1:n, 3 )
```

```
[1] 3 2 7 9 1 2 3 4 3
```

6.5.6. Reuso o reciclado

```
> v <- c( 3, 2, 7, 9 )
>
> # Reuso
> v * 1:2       # Reuso de elementos, cuando se acaba el vector se
```

```
[1]  3  4  7 18
```

```
>               # sigue la operación con el primero, el segundo,...
```

6.6. Matrices

Las matrices son conjuntos ordenados de vectores, así ahora, tenemos dos subíndices, el primero que hace referencia a la fila y el segundo a la columna.

6.6.1. Creación

```
> m <- matrix( 1:12, 3, 4 )     # Construir una matriz desde un vector
> m
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
> dim( m )                      # dimensiones de la matriz
```

```
[1] 3 4
```

```
> ncol( m )                     # Número de filas
```

```
[1] 4
```

```
> nrow( m )                     # Número de columnas
```

```
[1] 3
```

6.6.2. Creación otro ejemplo

```
> v <- c( 3, 2, 7, 9 )
> n <- length( v )
>
> m <- matrix( c( v, 1:n ), n ) # Unir dos vectores e una matriz
> dim( m )                      # dimensiones de la matriz
```

```
[1] 4 2
```

```
> ncol( m )                     # Número de filas
```

```
[1] 2
```

```
> nrow( m )                     # Número de columnas
```

```
[1] 4
```

6.7. Operaciones con matrices

```
> v <- c( 3, 2, 7, 9 )
> n <- length( v )
> m <- matrix( c( v, 1:n ), n )
>
> t( m )                        # Trasponer un matriz
```

```
     [,1] [,2] [,3] [,4]
[1,]    3    2    7    9
[2,]    1    2    3    4
```

```
> w <- c( 10, 1 )               # Vector de ponderación
> m %*% w                       # Producto matricial
```

```
     [,1]
[1,]   31
[2,]   22
[3,]   73
[4,]   94
```

6.7.1. Seleccionando filas o columnas

```
> v <- c( 3, 2, 7, 9 )
> m <- matrix( c( v, 1:length( v ) ), length( v ) )
>
> m[ 1,  ]           # La primera fila de la matriz
```

```
[1] 3 1
```

```
> m[ , 1 ]           # La columna fila de la matriz
```

```
[1] 3 2 7 9
```

```
> m[ 1:2,  ]         # Las dos primeras filas de la matriz
```

```
     [,1] [,2]
[1,]    3    1
```

```
[2,]    2    2
```

```
> m[ , 1:2 ]          # Las dos primeras columnas de la matriz
```

```
     [,1] [,2]
[1,]    3    1
[2,]    2    2
[3,]    7    3
[4,]    9    4
```

```
> m[ 1:2, 1:2 ]       # Las dos primeras filas y columnas
```

```
     [,1] [,2]
[1,]    3    1
[2,]    2    2
```

```
> m[ 1, -2 ]          # la primera fila sin la columna 2
```

```
[1] 3
```

6.8. *Data frames*

El principal problema de las matrices reside en la homogeneidad de los tipos de valores: todas las columnas debe ser homogéneas; sin embargo, los datos pueden tener distintos tipos de columnas, es para este caso, cuando se definen los *data frames* que simplemente tablas de datos.

```
> # Preparación de una tabla a partir de los datos de localidad
> # la abundancia de una especie y la cota de la localidad.
>
> tabla <- data.frame( row.names  = c( "S1", "B3", "L2", "M9" ),
+                      abundancia = c(    3,    9,    2,   1  ),
+                      cota       = c(  110,   76,   92,  83  ) )
> tabla               # Toda la tabla
```

```
   abundancia cota
S1          3  110
B3          9   76
L2          2   92
M9          1   83
```

6.8.1. Seleccionando elementos de *data frames*

```
> tabla <- data.frame( row.names  = c( "S1", "B3", "L2", "M9" ),
+                      abundancia = c(    3,    9,    2,   1  ),
+                      cota       = c(  110,   76,   92,  83  ) )
>
> tabla[ 1,   ]     # Primera fila de la tabla
```

```
   abundancia cota
S1          3  110
```

```
> tabla[  , 1 ]     # Primera columna de la tabla
```

```
[1] 3 9 2 1
```

```
> tabla$cota        # La columna de nombre localidad
```

```
[1] 110  76  92  83
```

```
> rownames( tabla ) # Nombres de las filas
```

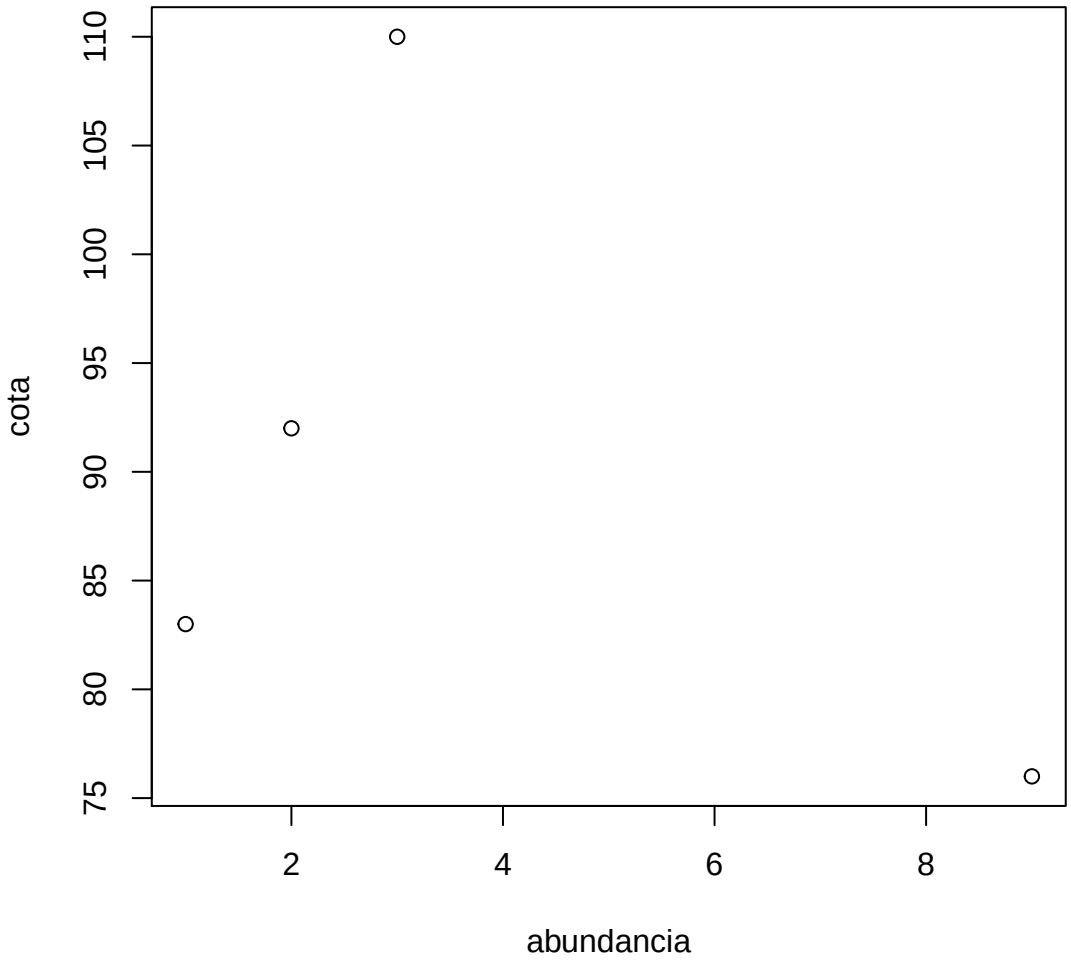
```
[1] "S1" "B3" "L2" "M9"
```

6.8.2. Notación alternativa con *data frames*

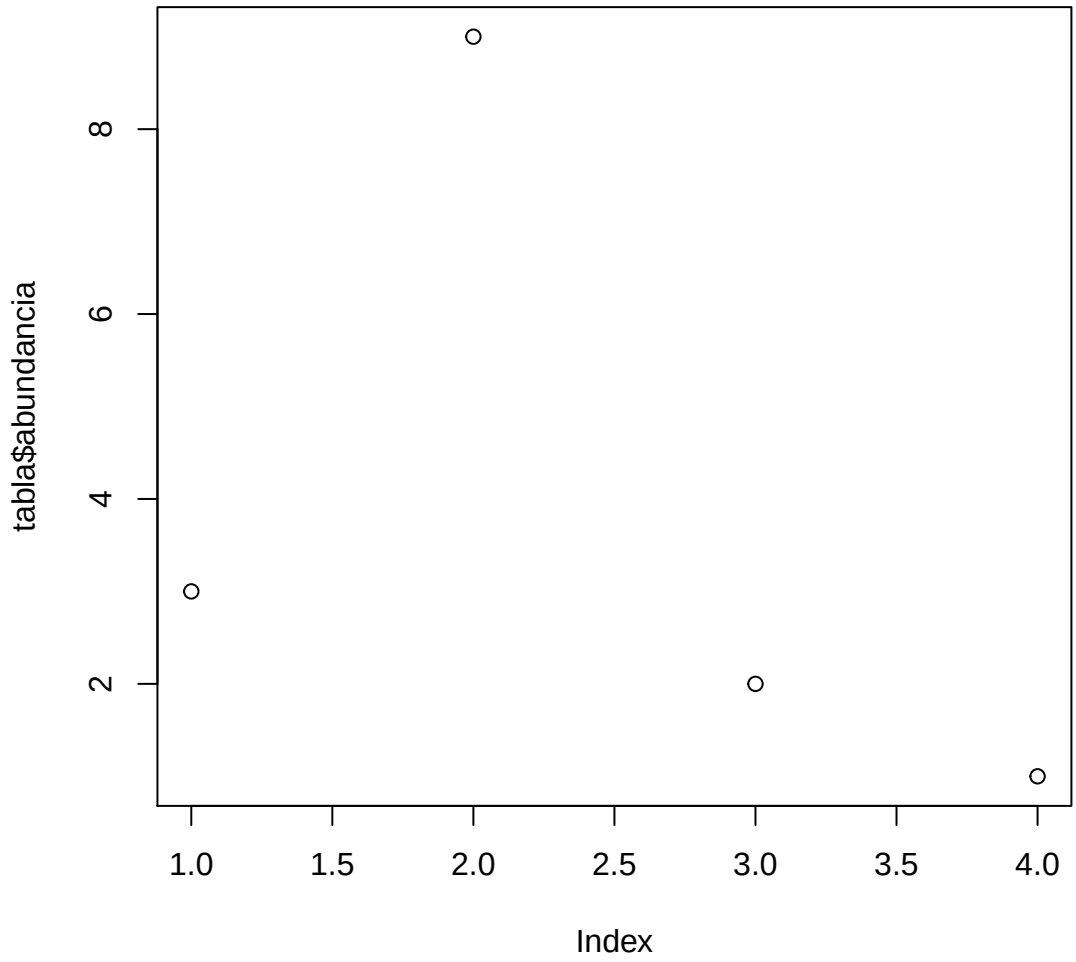
```
> tabla <- data.frame( row.names  = c( "S1", "B3", "L2", "M9" ),
+                      abundancia = c(    3,    9,    2,    1  ),
+                      cota       = c(  110,   76,   92,   83  ) )
>
> tabla$abundancia
```

```
[1] 3 9 2 1
```

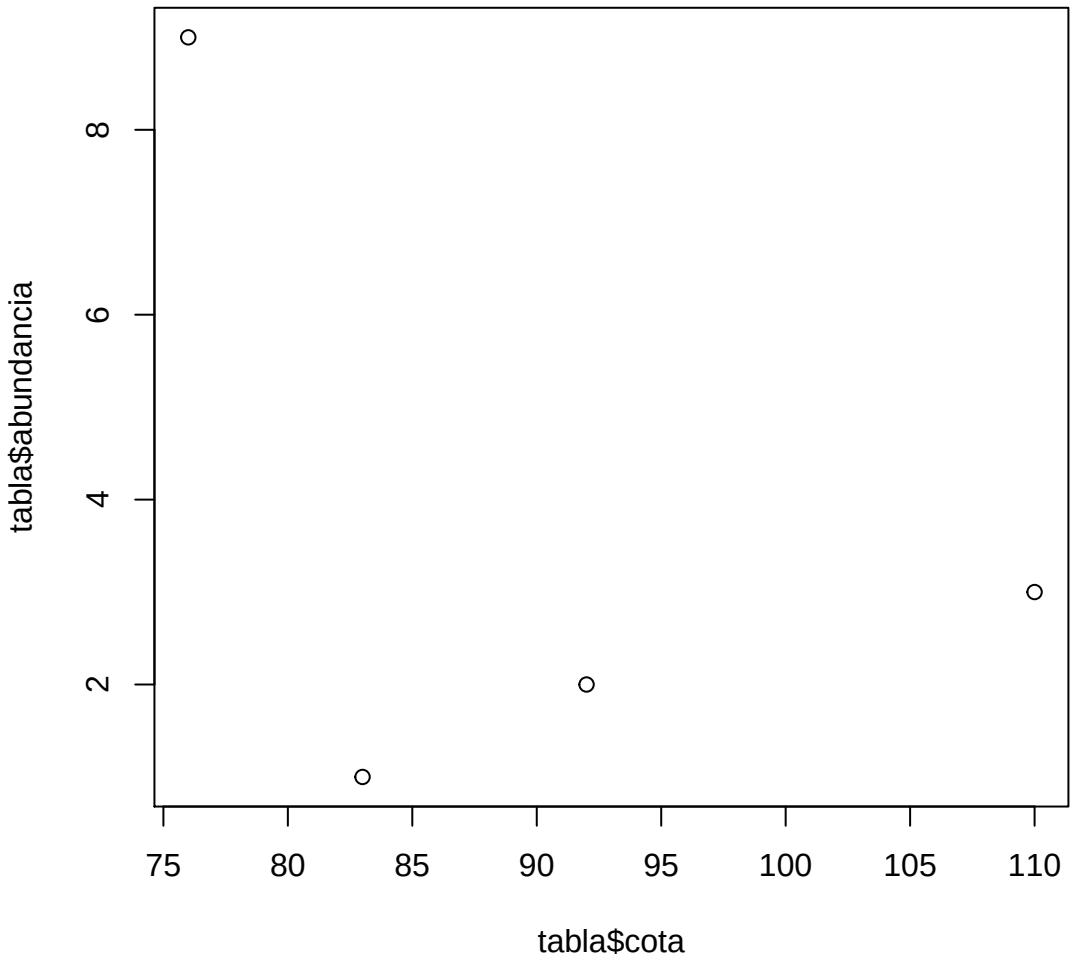
```
> plot( tabla )       # Representación gráfica de la tabla
```



```
> plot( tabla$abundancia )
```




```
> plot( tabla$cota, tabla$abundancia )
```



6.8.3. Leyendo datos desde fichero a *data frames*

```
> # Lectura de un fichero de datos
> biom <- read.table( "http://www.um.es/docencia/emc/datos/biom2003.dat" )
> names( biom )     # ¿Cuáles son los nombre de las columnas de la tabla?
```

```
 [1] "Grupo"   "Peso"    "Altura"  "Pie"     "Hombros" "Brazos"  "Caderas"
 [8] "Sexo"    "Ojos"    "Tipo"
```

```
> head( biom )      # Ver las primeras filas del objeto biom
```

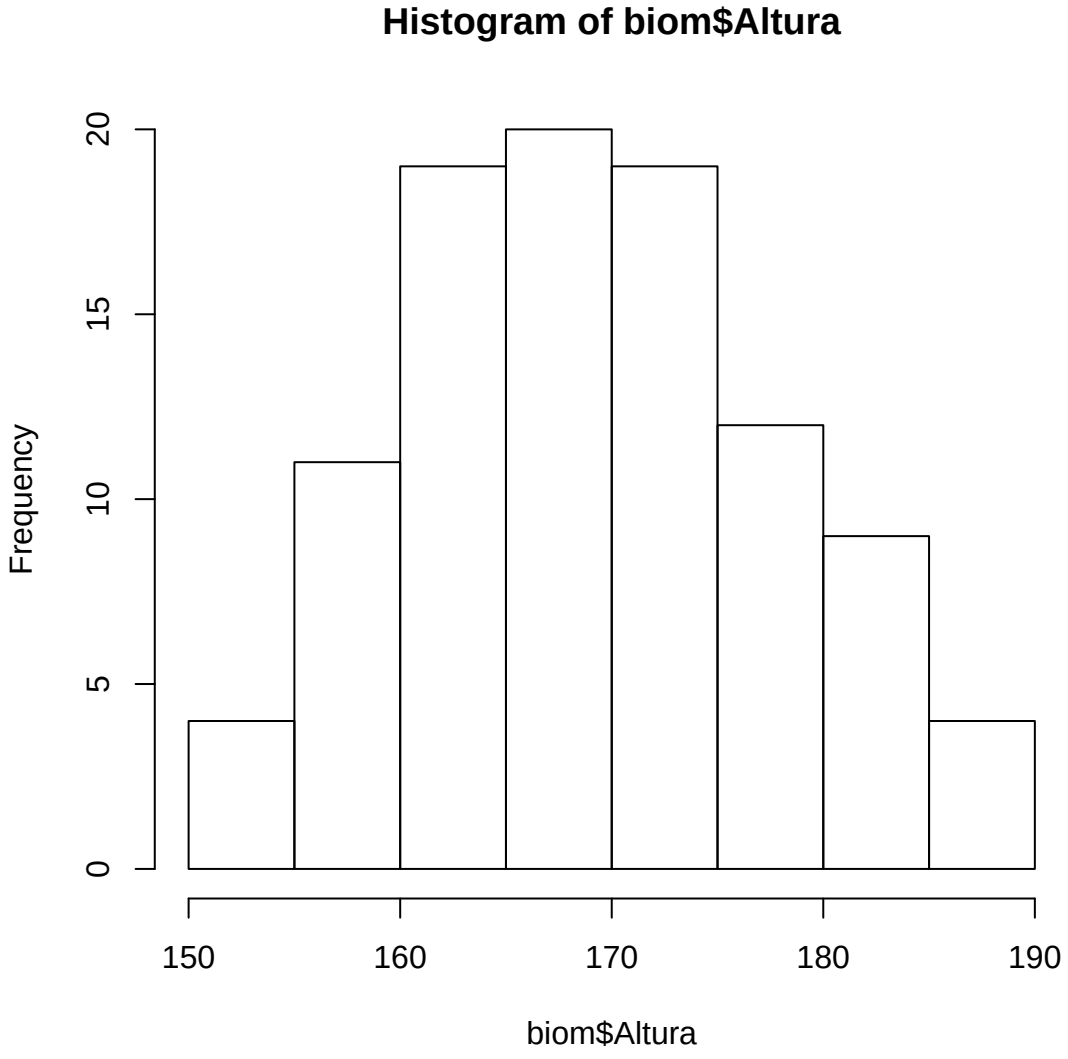
```
  Grupo Peso Altura Pie Hombros Brazos Caderas Sexo Ojos Tipo
1     1   60    163  37      41     68      95    1    1    2
2     1   52    166  37      37     70      87    1    2    1
3     1   61    172  39      39     69      91    1    2    1
4     1   73    181  43      50     78     101    2    2    1
5     1   53    172  39      39     72      89    1    1    1
6     1   63    169  40      37     66      96    1    2    1
```

6.9. Tipos de objetos: listas

En ocasiones necesitamos manejar conjuntos de elementos heterogéneos, vectores, matrices, ... en R estos conjuntos se denominan listas. Cada elemento tiene asignado un nombre y un tipo, a modo de ejemplo veamos como utilizar la lista que genera la función `hist`.

6.9.1. Creando una lista

```
> biom <- read.table( "http://www.um.es/docencia/emc/datos/biom2003.dat" )
>
> hist( biom$Altura )                    # histograma de los datos de altura
> histoAltura <- hist( biom$Altura )     # creación del objeto histograma de la altura
```




```
> names( histoAltura )                   # names del objeto
```

```
[1] "breaks"   "counts"   "density"  "mids"     "xname"    "equidist"
```

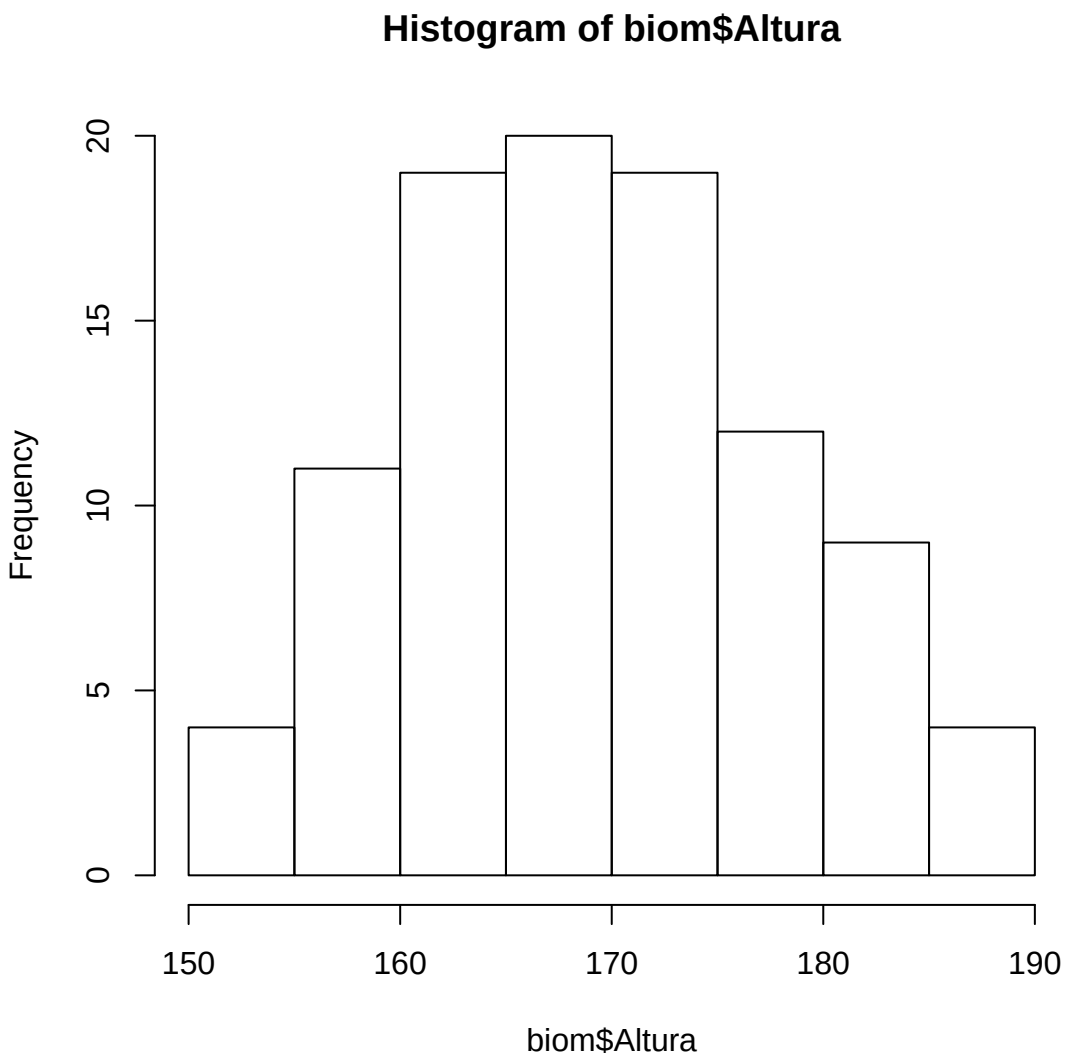
```
> str( histoAltura )                     # Estructura del objeto
```

```
List of 6
 $ breaks  : num [1:9] 150 155 160 165 170 175 180 185 190
```

```
 $ counts  : int [1:8] 4 11 19 20 19 12 9 4
 $ density : num [1:8] 0.00816 0.02245 0.03878 0.04082 0.03878 ...
 $ mids    : num [1:8] 152 158 162 168 172 ...
 $ xname   : chr "biom$Altura"
 $ equidist: logi TRUE
 - attr(*, "class")= chr "histogram"
```

6.9.2. Usando los elementos de una lista

```
> biom <- read.table( "http://www.um.es/docencia/emc/datos/biom2003.dat" )
> histoAltura <- hist( biom$Altura )
```



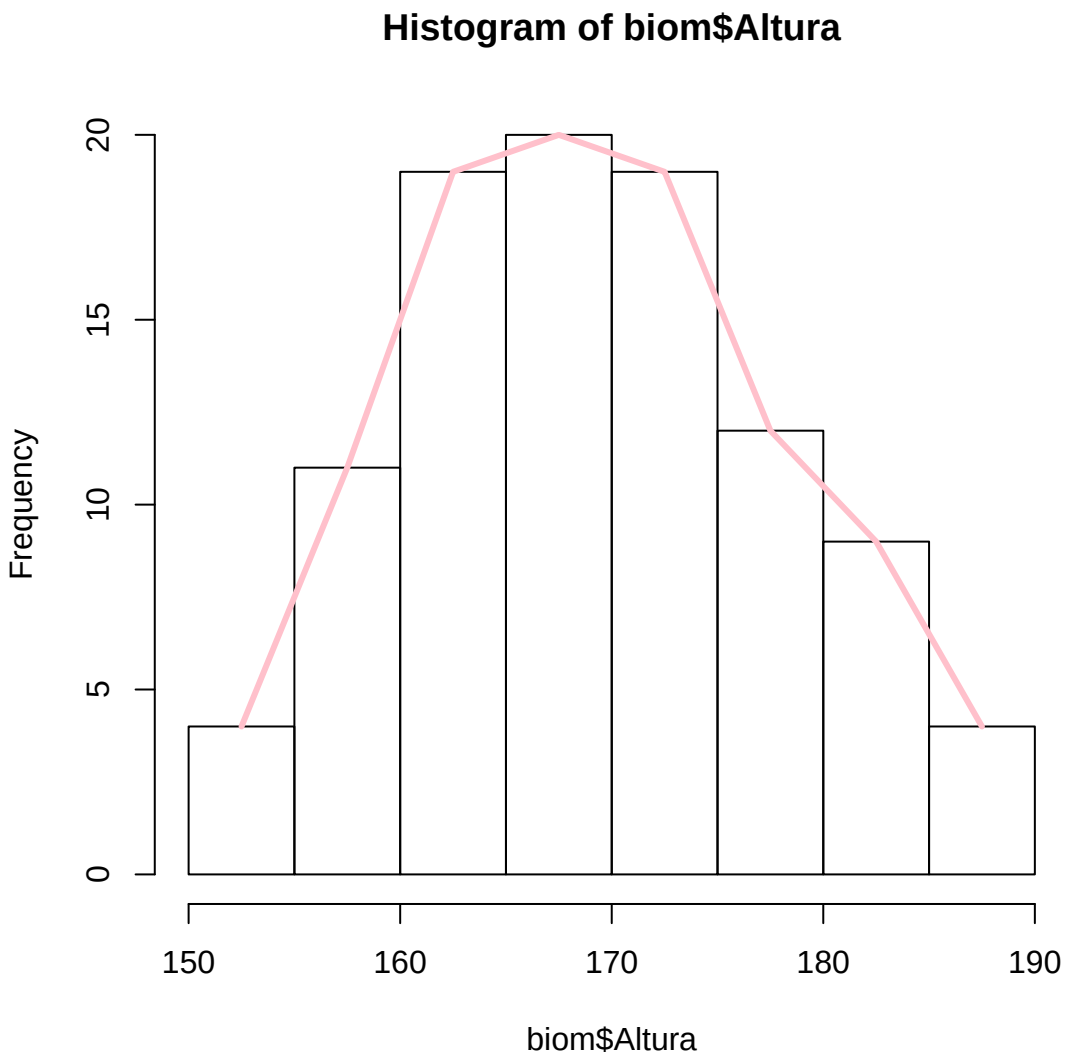

```
> histoAltura$counts                    # frecuencias las clases, contenidos en $counts
```

```
[1]  4 11 19 20 19 12  9  4
```

```
> histoAltura$mids                      # puntos medios de los intervalos de las clases
```

```
[1] 152 158 162 168 172 178 182 188
```

```
> plot( histoAltura )                    # representación del histograma
> lines( histoAltura$mids,
+        histoAltura$counts,
+        col = "pink", lwd = 3 )         # Polígono de frecuencias
```




```
# Datos y ficheros
```

6.10. Ficheros de datos

Como hemos visto en los ejemplos anteriores podemos tener datos en ficheros, en general, se trata de fichero en texto plano que pueden leerse con la simple ayuda del bloc de notas. Por ejemplo si guardamos el siguiente contenido en un fichero con un nombre legal podremos utilizarlo como un fichero de datos:

```
   x  y  z
1  6  4  A
2  9  3  A
3  2  1  B
```

Las líneas no deben iniciarse con espacios en blanco. No olvides que la última línea del fichero debe estar vacía, de lo contrario tendrás un feo mensaje en la pantalla, que no tiene efecto alguno en la lectura.

En los nombres de los ficheros:

- prohibidas las tildes (en el fichero y en los directorios)
- desaconsejados los espacios en blanco
- extensión: `.csv` o `.txt` o `.dat`
- se recomienda uso de `localc` como "editoR de datos" y para el manejo de ficheros `.csv`

Si lo llamamos `datos.csv` y lo guardamos en nuestro directorio personal, entonces podremos leerlo sin problemas.

6.11. Lectura de ficheros

Los directorios albergados en nuestro disco duro puede ser cargados en R si cumplen las condiciones indicadas y sabemos donde se ubican, el resultado es un *data frame*.

1. En el directorio de trabajo

```
> getwd()                    # determinar el directorio de trabajo
```

```
[1] "/home/amaurandi/Git/007cursodocint/1617/INTR/www-portal/docs"
```

```
> # setwd( "c:/git/directorioProyecto" ) #
> # read.table( "datos.dat" ) # lectura de fichero en el directorio de trabajo
```

1. En el directorio personal u otro subdirectorio

```
> # read.table( "~/datos.dat" )
```

\

```
> # read.table( "~/proyecto/muestreoCampoEnero/datos.dat" ) # lectura de fichero en el directorio de tr
```

1. En servidores con acceso web

```
> # lectura de un fichero disponible en un servidor
> biom <- read.table( "http://www.um.es/docencia/emc/datos/biom2003.dat" )
> names( biom )
```

```
 [1] "Grupo"   "Peso"    "Altura"  "Pie"     "Hombros" "Brazos"  "Caderas"
 [8] "Sexo"    "Ojos"    "Tipo"
```