· Ejercicio: leer 2 ficheros de sonido (WAV o MP3)

Tras instalar RStudio debemos instalar desde la consola las librerías que nos permiten trabajar con estos formatos de sonido. En nuestro caso instalaremos las 2 siguientes:

```
install.packages('tuneR', dep=TRUE)
install.packages('seewave', dep=TRUE)
```

Ahora tocaría indicar el directorio donde se encuentran los sonidos, el directorio del reproductor de Windows y cargar y abrir los archivos de sonido correspondientes.

```
# Cargamos las librerias
library(tuneR)
library(seewave)
library(audio)

# Indicamos el directorio donde se encuentran los sonidos
setwd("C:/Users/Alberto/Desktop/PDIH/PRACTICAS/P4/S6-sounds")

# Indicamos el directorio del reproductor de windows
# ERROR -> setWavPlayer("C:/Program Files/Windows Media Player/wmplayer.exe")
system(paste('"C:/Program Files/Window Media Player/wmplayer.exe"' , 'mysong.wav'))

# Cargamos y reproducimos
hola <- readWave('hola.wav')
hola
listen(hola)

perro <- readWave('perro.wav')
perro
listen(perro)

intro20th <- readMP3('intro20th.mp3')
intro20th
listen(intro20th)
```

· Ejercicio: dibujar la forma de onda de los sonidos

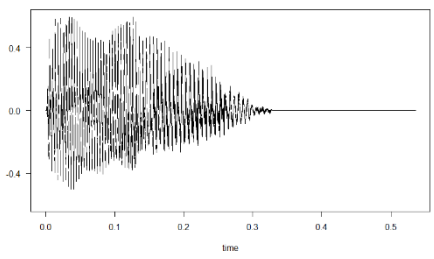
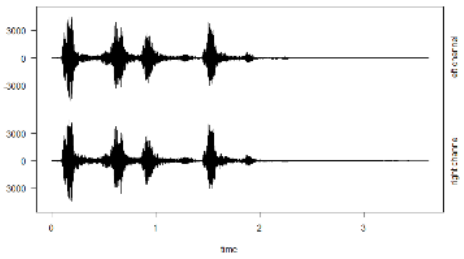
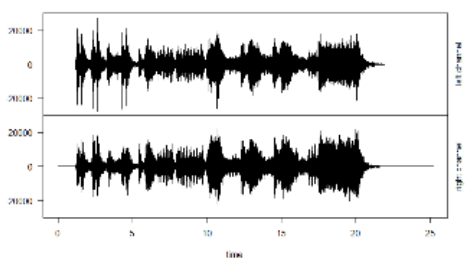
Para obtener el 'plot' de la onda correspondiente a cada uno de los sonidos anteriormente leidos basta con lanzar el siguiente comando visto en el seminario.

```
# Forma de onda de 'hola.wav'
plot(extractWave(hola, from=1, to=11809))

# Forma de onda de 'perro.wav'
plot(extractWave(perro, from=1, to=159732))

# Forma de onda de 'intro20th.mp3'
plot(extractWave(intro20th, from=1, to=1110528))
```

Las ondas resultantes mostradas en orden serían las siguientes:

· Ejercicio: obtener la información de las cabeceras de los sonidos

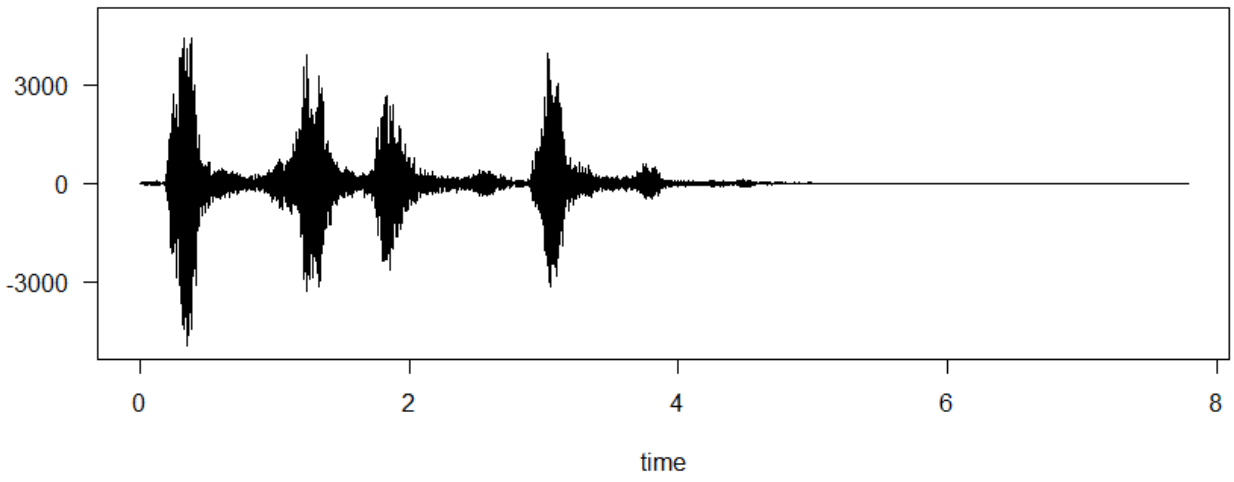
La salida por consola seria la siguiente, aunque también se podría consultar directamente dicha información en la pestaña “Environment” apartado “Data”.

```
> str(hola)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : num [1:11809] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ right    : num(0)
  ..@ stereo   : logi FALSE
  ..@ samp.rate: int 22050
  ..@ bit      : int 32
  ..@ pcm      : logi FALSE
> str(perro)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:159732] 0 0 0 0 0 0 0 0 1 1 ...
  ..@ right    : int [1:159732] 0 0 0 0 0 0 0 0 1 1 ...
  ..@ stereo   : logi TRUE
  ..@ samp.rate: int 44100
  ..@ bit      : int 16
  ..@ pcm      : logi TRUE
> str(intro20th)
Formal class 'Wave' [package "tuneR"] with 6 slots
  ..@ left     : int [1:1110528] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ right    : int [1:1110528] 0 0 0 0 0 0 0 0 0 0 ...
  ..@ stereo   : logi TRUE
  ..@ samp.rate: num 44100
  ..@ bit      : num 16
  ..@ pcm      : logi TRUE
```

· Ejercicio: unir varios sonidos en uno nuevo

```
# Union de sonidos
union <- pastew(hola,perro,output="Wave")
union
listen(union)
```

· Ejercicio: dibujar la onda resultante de la unión de sonidos



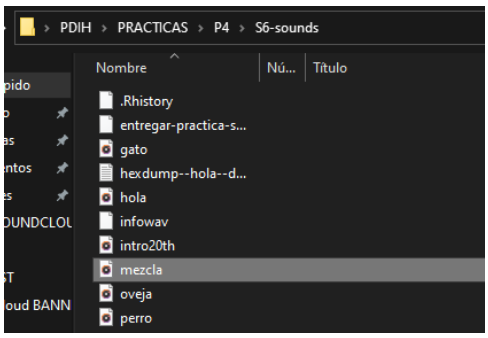
· Ejercicio: pasarle un filtro de frecuencia para eliminar aquellas por debajo de los 3.000Hz

```
# Eliminar frecuencias por debajo de los 3000Hz
f <- union@samp.rate
resultado <- bwfilter(union, f=f, channel=1, n=1, from=1, to=3000, bandpass=TRUE)
listen(resultado,f=f)
```

· Ejercicio: almacenar la señal obtenida como "mezcla.wav"

Una vez hemos terminado la edición de un sonido podemos guardar el resultado en disco en el formato deseado de la siguiente manera:

```
# Guardar la edicion anterior en disco en formato WAV
writeWave(resultado, file.path("mezcla.wav") )
```



· Ejercicio: cargar un nuevo archivo de sonido, aplicarle eco y a continuación darle la vuelta al sonido. Almacenar la señal obtenida como un fichero WAV denominado "alreves.wav".

```
# Cargar audio, aplicarle eco, darle la vuelta a dicho sonido y almacenarlo en disco
oveja <- readWave('oveja.wav')
oveja
listen(oveja)
f<-oveja@samp.rate
eco_oveja <- echo(oveja, f=f, amp=c(0.8,0.4,0.2), delay=c(1,2,3), output="wave")
listen(eco_oveja)
oveja_del_reves <- revw(eco_oveja, output="wave")
writeWave(oveja_del_reves, "oveja_del_reves.wav")
listen(oveja_del_reves)
```