USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

Módulo 9

Comunicación con R

Alfredo Rigalli
Maela Lupo

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

2023

ISBN 978-987-86-3474-6

USO DE HERRAMIENTAS INFORMÁTICAS PARA LA RECOPILACIÓN, ANÁLISIS E INTERPRETACIÓN DE DATOS DE INTERÉS EN LAS CIENCIAS BIOMÉDICAS

MODULO 9

Comunicación con R

Alfredo Rigalli
Maela Lupo

Centro Universitario de Estudios Medioambientales
Facultad de Ciencias Médicas
Universidad Nacional de Rosario

Uso de herramientas informáticas para la recopilación, análisis e interpretación de datos de interés en las ciencias biomédicas : módulo 9 : Comunicación con R / Alfredo Rigalli ... [et al.]. - 1a edición para eel alumno - Rosario : Alfredo Rigalli, 2020.
Libro digital, PDF

Archivo Digital: online
ISBN 978-987-86-3474-6

1. Matemática Aplicada. 2. Matemática Estadística. I. Rigalli, Alfredo
CDD 510

AUTORES

Lupo, Maela: Licenciada en biotecnología y Doctora en Ciencias Biomédicas. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario.

Rigalli, Alfredo: Bioquímico y Doctor en bioquímica. Investigadora del Laboratorio de Biología Ósea y del Centro Universitario de Estudios Medioambientales y docente de la cátedra de Química Biológica de la Facultad de Ciencias Médicas de la Universidad Nacional de Rosario. Investigador independiente del Consejo de Investigaciones de la UNR y del CONICET

Tabla de contenidos

ORGANIZACIÓN DE LA OBRA

Esta obra está dividida en módulos y clases. Cada módulo agrupa temas diferentes. Brevemente
Módulo 1: Instalación e introducción a R
Módulo 2: gráficas con R
Módulo 3: introducción a la estadística básica.
Módulo 4: análisis multivariado de datos numéricos y análisis especiales de datos.
Módulo 5: desarrollo de scripts y programación en R.
Módulo 6: Matemática con R
Módulo 7: Modelización matemática con R
Módulo 8: Aprendizaje automatizado
Cada módulo se divide en 9 clases, las cuales constan de tablas específicas para cada clase, así como de un vídeo y una ejercitación. Al final de las 9 clases existe un examen final del módulo.
Las planillas de cálculo en formatos ods o xls llevarán la denominación tablaR1-3.ods por ejemplo si es la planilla de cálculo para la clase 3 del módulo 1. En el interior de la planilla hallará tablas con los nombre tablaR131, tablaR132, tablaR133, etc. Todas tablas para el módulo 1 (primer número), de la clase 3 (segundo número) y el tercer número indica el número de tabla. Con estos nombres serán introducidos como objetos en el espacio de trabajo.
Al principio de la obra hallará los links a un vídeo sobre la clase y tendrá un link a la planilla de cálculo con las tablas para el desarrollo de la clase.
Resaltado en este color hallará los textos que representan códigos de R. A su vez estos códigos se identifican porque comienzan con el caracter ">" . Estos códigos puede copiarlos y ejecutarlos. Para ello no copie el caracter ">".

Links videos para desarrollo de clases

Clase 1

vídeo: https://youtu.be/QxkWoPjtxiw

Clase 2

vídeo: https://youtu.be/IcpSbnO-4Hc

Clase 3

vídeo: https://youtu.be/KGRrVFn9bJI

Clase 4

Video: https://youtu.be/OO7-0Bx1nTw

Clase 5

Video: https://youtu.be/COwa34AY3sw

Clase 6

Video: https://youtu.be/bjTrdMDwlyI

Clase 7

Vídeo: https://youtu.be/pvU8HeUybEk

Clase 8

Vídeo: https://youtu.be/tdpwoUKBxQQ

Clase 9

Vídeo: https://youtu.be/a-ckLlUtR0c

Módulo 9 – clase 1

Uso de R en modo multiusuario

Lo común es que R sea utilizado en una computadora por un usuario para realizar almacena los datos y realiza el análisis de datos para arribar a conclusiones. Estas actividades normalmente son resueltas con bibliotecas disponibles en los repositorios, utilizando funciones clásicas. Los usuarios con mayor expertisia asisten sus trabajos con scripts propios o diseñados por otros, construyen sus propias funciones y paquetes. Como se desarrolló en el módulo 5 de este curso, R puede ser utilizado para crear un MegaScript que llegue a administrar un laboratorio, por ejemplo.
A un script de este tipo accederán personas que trabajen en ese laboratorio a través de una única PC en el más sencillo de los casos o bien a través de computadoras en red, donde las baes de datos de R y el script se encuentra en el servidor. Los usuarios pueden acceder desde computadoras cliente, utilizando diferentes mecanismos de comunicación.
Cuando son más de un usuario los que pueden acceder a una misma base de datos, se presenta el fenómeno de concurrencia, que si no es adecuadamente manejado puede conducir a errores de almacenamiento de información y funcionamiento del sistema.
¿Que es la concurrencia en ciencias de la computación? La concurrencia es la posibilidad de ejecutar diferentes partes de un sistema de manera simultánea sin afectar su funcionamiento. Si no se toman los recaudos necesarios, esto puede llevar a errores de almacenamiento y procesamiento de datos, a los que llamamos errores de concurrencia.
Si bien el problema puede ser muy grande y las soluciones muy sofisticadas, mostraremos algunos ejemplos sencillos para comprender el problema y como solucionarlo.

Acceso a una base de datos por dos usuarios simultáneamente

El objetivo de este ejemplo es visualizar el problema que puede ocasionar la concurrencia. Supongamos que tenemos un data.frame que almacena el nombre de cada integrante de un laboratorio y la fecha y hora de ingreso. Diseñamos para eso un script que cada usuario ejecuta desde una dada computadora al llegar al trabajo.
Así, cada integrante al llegar al laboratorio, ejecuta los siguientes pasos
1- abre una consola en el directorio correspondiente, en este ejemplo lo colocaremos en **Home/Desktop/R91**
2- ejecuta el script **usuario.R**, donde deberá completar sus datos y el script automáticamente crea un nuevo registro en el data.frame() al que le daremos el nombre **usuariollegada**
3- sale de R, guardando el espacio de trabajo
4- cierra la consola
5- se va a trabajar
Cuando llegue el siguiente integrante del laboratorio repetirá los pasos anteriores.
Así en el data.frame irán quedando registrado en cada línea los sucesivos integrantes del laboratorio. Será un buen mecanismo de control! Lo tenemos implementado en el CUEM y da muy buen resultado!!!
Creamos primero el espacio de trabajo para almacenar esta "aplicación". Para ello abrimos R en Home/Desktop/R91
En el espacio de trabajo creamos un data.frame() que llamamos **usuariollegada**. Puede utilizar cualquier mecanismo que conozca, pero lo platearemos como se hizo en el módulo 1.
Creamos un data.frame vacío

```
> usuariollegada<-data.frame()
```

lo editamos

```
> usuariollegada<-edit(usuariollegada)
```

colocamos a las dos columnas que necesitamos los nombres: fecha y nombre, ambos en formato character. Introducimos un primer registro con una fecha 1/1/1 y nombre NN, Figura 1.

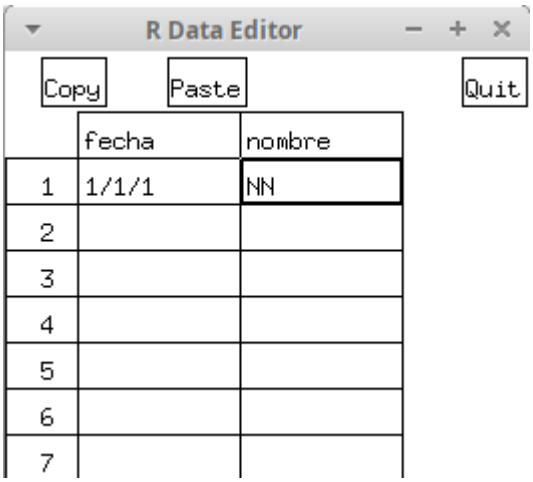


Figura 1

Al oprimir Quit, nos quedará el data.frame con el primer registro. Podemos chequear que así sea

```
> usuariollegada
```

```
  fecha nombre
1 1/1/1     NN
```

Vemos ya el primer registro que hemos introducido al crear el data.frame.

Diseñamos un sencillo script para registrar la asistencia por el usuario. En la tabla siguiente tenemos el script a la izquierda y su explicación a la derecha

usuario.R	explicaciones
print('Buen día, introduzca apellido y nombre') nombre<-as.character(scan(file = "", what = "",nmax=1,sep="\n",nlines=1)) filas<-nrow(usuariollegada) usuariollegada[filas+1,1]<-nombre usuariollegada[filas+1,2]<-as.character(date()) print('Su ingreso fue registrado')	Muestra un mensaje en pantalla solicitando el nombre Una vez introducido el nombre lo asigna a la variable nombre cuenta las filas del data.frame usuariollegada en la fila siguiente columna 1 coloca el nombre en la misma fila pero columna 2 coloca la fecha le informa al usuario que los datos fueron registrados.

Copie el código de la columna de la izquierda, péguelo en un porcesador de textos y guardelo con el nombre usuario.R, en la carpeta R91, que acaba de crear

Veamos funcionamiento
Supongamos que hacemos funcionar el sistema desde consola. Un mecanismo sería que el usuario se posicione en el directorio R91 del Desktop, inicie una consola.
Ejecuta el script usuario.R con

```
> source('usuario.R')
```

carga su nombre y apellido
Así quedará incluido en el data.frame usuariollegada su nombre apellido y fecha
Pide salir de R
sale guardando el espacio de trabajo

Cuando venga otro usuario repetirá los pasos anteriores y así quedarán guardados en orden cronológico cada uno de los integrantes. Pruebe varios ingresos y luego compruebe los mismos pidiendo el data.frame

> usuariollegada

Tendrá el listado de los ingresos realizados
Algo tan sencillo, aunque efectivo, parece no tener forma de fallar. Pero como dice un viejo refrán: Todo tiene una sola forma de andar y 100 formas de fallar.
¿Que ocurrirá si un usuario deja la consola abierta sin guardar? Es una acción bastante común.
Hasta que no guarde el espacio de trabajo, esos datos no quedarán registrados. Si deja la consola abierta (quizás minimizada), los datos estarán en la memoria volátil de la máquina, pero no guardados en las unidades de almacenamiento.
Si luego viene otro usuario y no se anoticia que hay una consola abierta, abre otra consola, ejecuta el script, carga sus datos, sale del espacio de trabajo, quedando sus datos guardados. Hasta aca todo bien. ¿Pero que ocurrirá cuando se cierre la otra consola que está en el mismo espacio de trabajo?
Al cerrarse la consola del primer usuario, se guardarán los datos de éste, sobreescribiendo el espacio de trabajo que abrió el segundo y por dicha razón el segundo usuario no quedará registrado.

Veamos el paso a paso
1- Llega Juan Perez al trabajo, abre la consola, ejecuta el script usuario.R, cargas sus datos pero no sale del espacio de trabajo, por lo cual es espacio de trabajo no quedó guardado, aunque los datos si estan cargados están en el data.frame, Figura 2

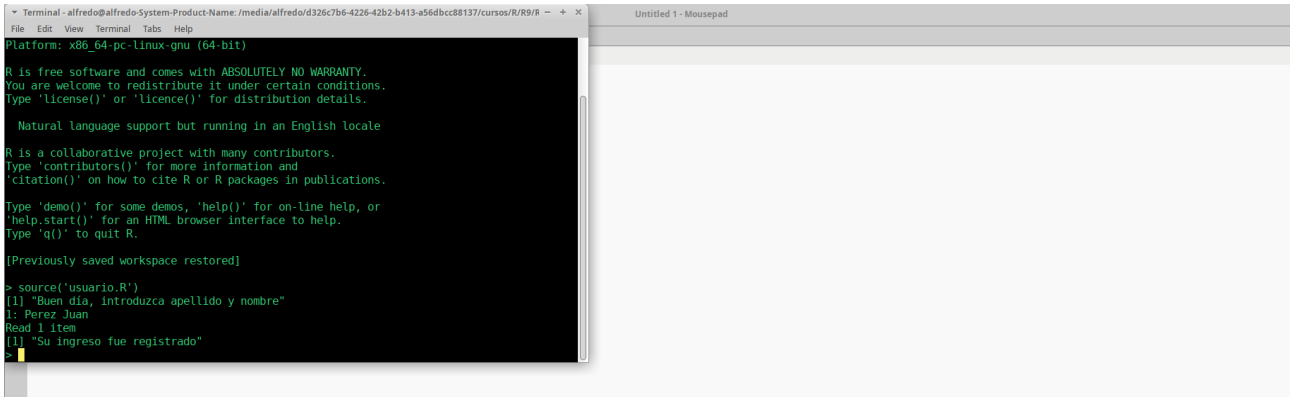


Figura 2

2- Llega Ana Ruiz y abre otra consola, carga R, ejecuta el script usuario.R y carga su datos. Vemos esto en la consola de la derecha de la Figura 3

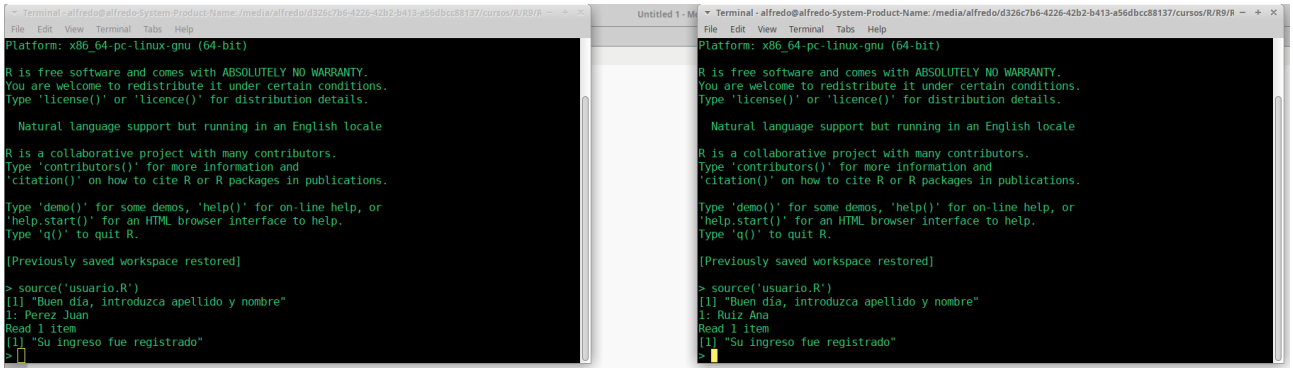


Figura 3
ambos usuarios cargaron sus datos. Como podemos ver en ambas consolas, al pedir el dataframe usuariollegada, vemos los ingresosm Figura 4

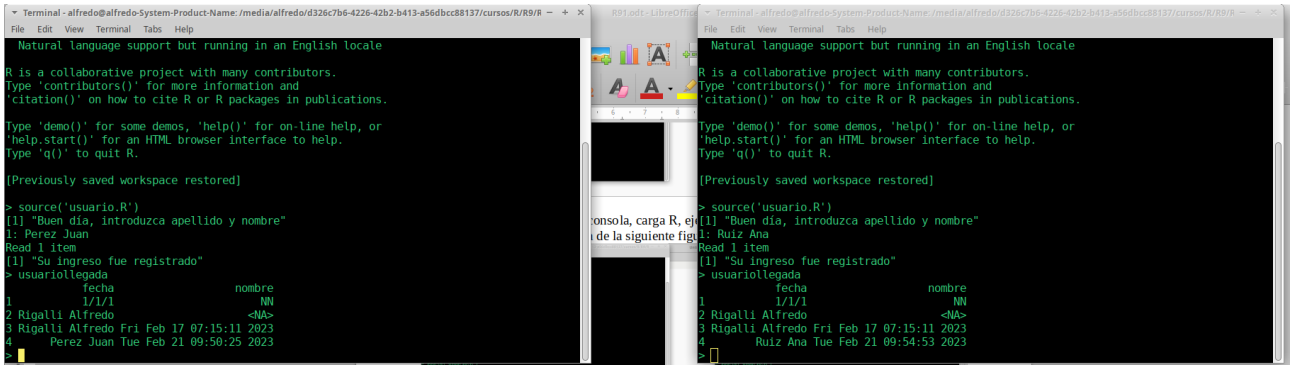


Figura 4

A la izquierda esta el ingreso de Juan Perez, y a la derecha el ingreso de Ana Ruiz.

3- Ana Ruiz, sale del espacio de trabajo y graba sus datos, para ello ejecuta

```
> q()
```

y elige la acción "y" para salir, Figura 5,

Ana se queda tranquila que sus datos de ingreso al trabajo han sido computados. Feliz y contenta se va a trabajar por su buena paga diaria. Queda solo la consola abierta por Juan Perez, Figura 6.

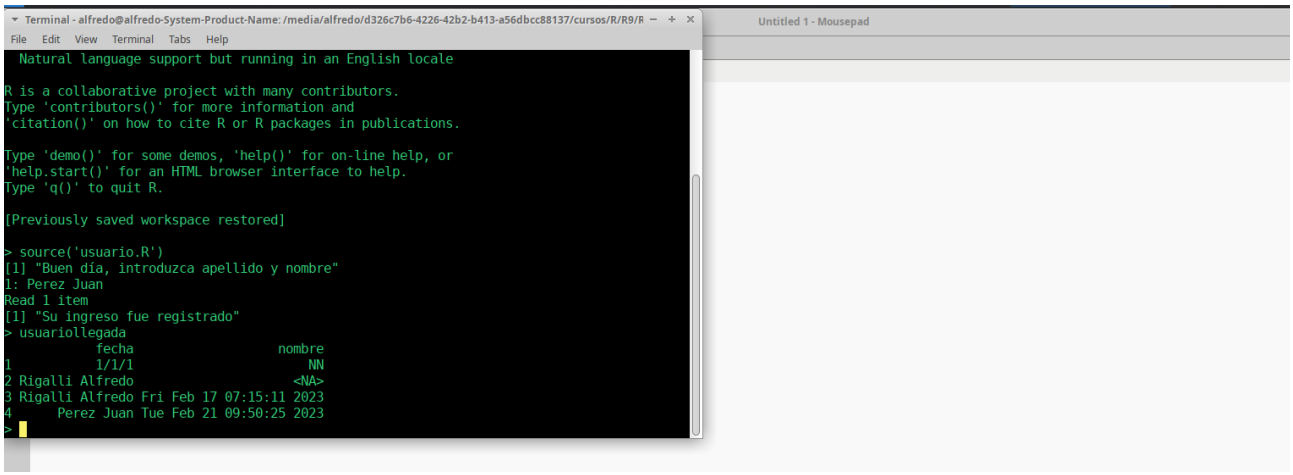
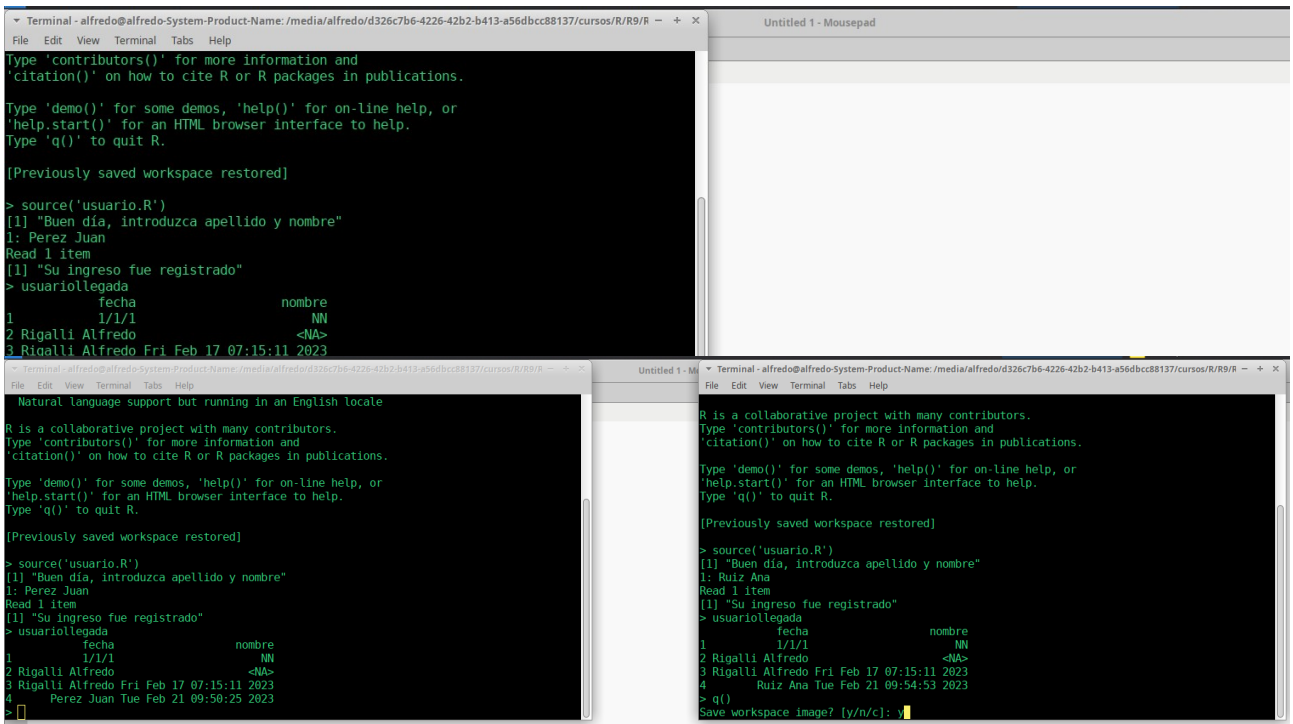


Figura 6

4- Juan Perez se acuerda que no guardó el espacio de trabajo ni cerró R. Vuelve a la PC, cierra R, toma la opción "y" para que guarde los datos. Con los que se asegura que "SUS" datos queden guardados. Ya que está cierra la consola con el comando exit, Figura 7.



Ahora todo parece estar bien Juan Perez registró su ingreso y lo guardo. Ana Ruiz, registró su ingreso y lo guardo.

Veamos si es cierto. El jefe del personal abre el espacio de trabajo y pide el contenido del data.frame: usuariollegada, que como podemos ver está presente entre los objetos, Figura 8.

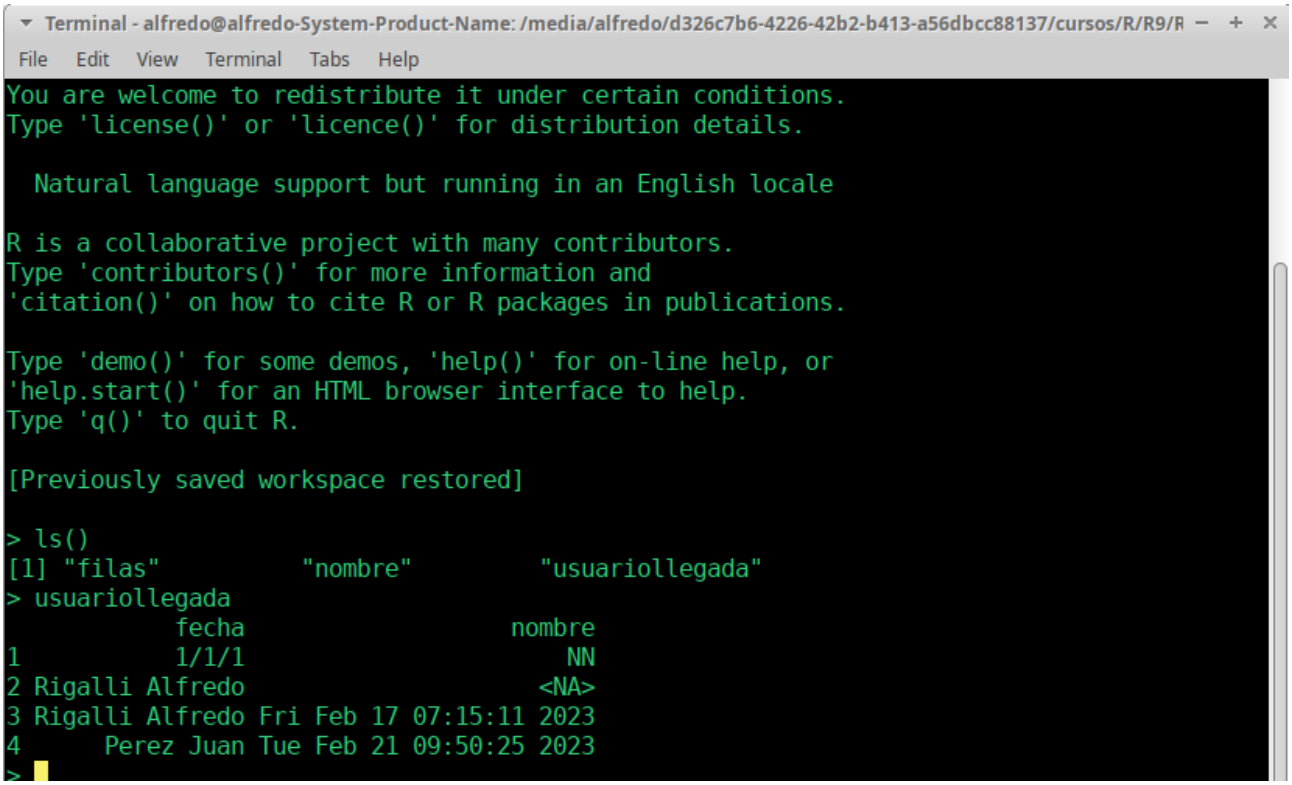


Figura 8

Como podemos ver solo está Juan Perez, como último ingreso.
¿Qué ocurrió? Si Ana Ruiz hizo todo perfecto!
Es cierto que Ana Ruiz hizo todo perfecto, pero Juan Perez tenía abierto el data.frame usuariollegada con anterioridad a la llegada de Ana, y por ende Ana no figuraba. Al cerrar R, se guarda ese data.frame y lamentablemente Ana a fin de mes notará el descuento de un día de trabajo, que había cumplido feliz y alegremente.
La concurrencia a una base de datos sin los recaudo adecuado ha conducido a un problema.
¿Quien hizo las cosas mal? ¿Juan Pérez? NO. Entonces fue Ana Ruiz? Tampoco. El problema es que está mal el script que permite un acceso simultáneo a una base de datos sin tomar los recaudos necesarios para evitar este tipo de errores.

Solución al problema

Si bien podrían pensarse varias soluciones diferentes con mayor o menor grado de sofisticación. veremos algunas que pueden ser de utilidad y que podremos entender e implementar de ser necesario con los conocimientos adquiridos en los primeros 8 módulos del curso.

Solución 1

Para este caso que solo tiene que cargar su apellido y luego salir. Agregar en el script una salida automática de R, grabando los datos. Además haremos que le avise al usuario que todo está bien y que en 5 segundos saldrá automáticamente. En la tabla siguiente vemos el script que llamaremos scriptautomatico.R, que tiene algunas modificaciones al script anterior

scriptautomatico.R	explicaciones

print('Buen día, introduzca apellido y nombre')
nombre<-as.character(scan(file = "", what = "",nmax=1,sep="\n",nlines=1))
filas<-nrow(usuariollegada)
usuariollegada[filas+1,1]<-nombre
usuariollegada[filas+1,2]<-as.character(date())
print('Su ingreso fue registrado')
print('No tiene más nada que hacer, en 5 segundos se cerrará R')
for(a in 6:1){
print(paste("tiempo remanente: ",a=a-1))
Sys.sleep(1)
}
quit(save = "yes", status = 0, runLast = TRUE)
	Muestra un mensaje en pantalla solicitando el nombre.
	Una vez introducido el nombre lo asigna a la variable nombre.
	Cuenta las filas del data.frame usuariollegada.
	En la fila siguiente columna 1 coloca el nombre
	En la misma fila pero columna 2 coloca la fecha
	Informa al usuario que los datos fueron registrados.
	Le avisa que no tiene que hacer nada más, R lo hará.
	Inicia un bucle que hara una cuenta regresiva en segundos.
	
	Sale automáticamente de R y guarda el espacio de trabajo

Para el caso planteado, sin duda es la mejor opción. Además si hemos creado el acceso directo al script podemos programar también que se cierre directamente la consola. Con este script el tiempo de permanencia es muy bajo en la base de datos y es poco probable que exista concurrencia. Sin embargo, si un usuario abre el script pero se demora entre que carga su apellido y nombre y oprimir enter, puede haber problemas. Desde otra terminal otro usuario podría hacerlo completando todo en poco tiempo y aparecería nuevamente el problema descripto. La solución planteada es mejor que nada, pero puede ser superada.

Solución 2

Impedir el acceso mientras el espacio de trabajo esté abierto. Esto se puede hacer de diversas maneras, pero lo más sencillo es que cuando un usuario abre el espacio de trabajo se cree un archivo, que bien puede ser un archivo de texto, en el mismo o en otro directorio. Llamaremos de aca en adelante bloqueo.txt a este archivo. Si otro usuario ingresa, lo primero que hace el script es fijarse si está el archivo de texto creado. En caso que exista, no le permite el ingreso. En caso contrario, si el archivo no está en el directorio, sí puede ingresar y completar sus datos. Cuando sale del espacio de trabajo, se borra el archivo de texto. Este archivo actua como una traba que se activa al ingresar al espacio de trabajo impidiendo el ingreso de cualquier otro usuario. Al salir del espacio de trabajo se borra el archivo, depejando el ingreso a otro usuario.
La figura siguiente muestra el recurso explicado anteriormente

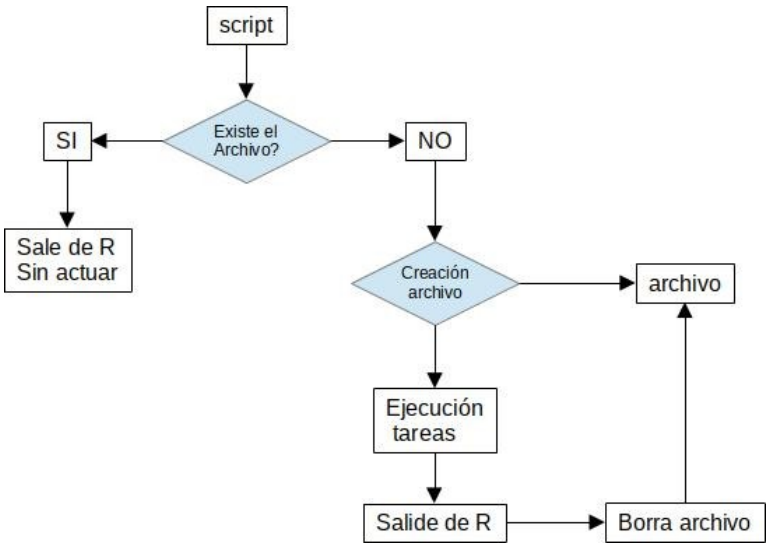


Figura 9

A continuación se muestra el script que realiza este procedimiento

usuariobloqueo.R	explicaciones

options(warn=-1)
if(file.exists('bloqueo.txt')==TRUE){
print('No puede acceder momentaneamente a la base, intentelo en unos minutos')
Sys.sleep(3)
quit(save = 'no', status = 0, runLast = FALSE)
}else{
file.create('bloqueo.txt')
}
options(warn=0)
print('Buen día, introduzca apellido y nombre')
nombre<-as.character(scan(file = "", what = "",nmax=1,sep="\n",nlines=1))
filas<-nrow(usuariollegada)
usuariollegada[filas+1,1]<-nombre
usuariollegada[filas+1,2]<-as.character(date())
print('Su ingreso fue registrado. Se cerrará R automáticamente')
file.remove('bloqueo.txt')
quit(save = 'yes', status = 0, runLast = FALSE)
	Elimina posibles warnings. Vea su uso en módulo 5
	Chequea si existe en el directorio el archivo bloqueo.txt
	Si existe nos da un mensaje que no puede acceder
	Da 3 segundos para ver el mensaje
	Sale de R
	Si no encuenta el archivo
	lo crea con la funcion file.create() dandole el nombre bloqueo.txt
	Permite nuevamente los warnings
	Pide que introduzca el nombre
	lo asigna a a variable nombre
	cuenta las filas actuales del data.frame usuariollegada
	asigna a columna 1 el nombre
	asigna a columna 2 la fecha
	Avisa al usuario de su registo correcto
	borra el archivo bloqueo.txt
	Se cierra R automáticamente

Veamos su funcionamiento.
En la Figura 10 siguiente se muestra una consola en la que ingresó el usuario y está presto a ejecutar el scritp usuariobloqueo.R, para registrar su asistencia. En la ventana de la derecha vemos el directorio que contiene al espacio de trabajo. Como podemos observar en el directorio no existe el archivo bloqueo.txt

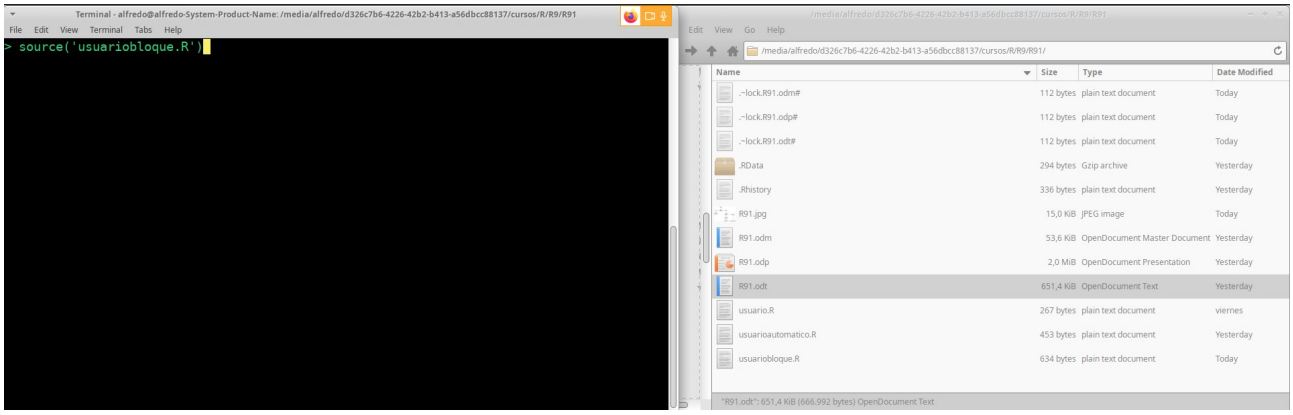


Figura 10

Si el usuario oprime enter, ejecutando el script, como no existe el archivo bloqueo.txt, podrá ingresar y el script le pedirá el nombre, pero como puede ver ha creado en el directorio al archivo bloqueo.txt, Figura 11.

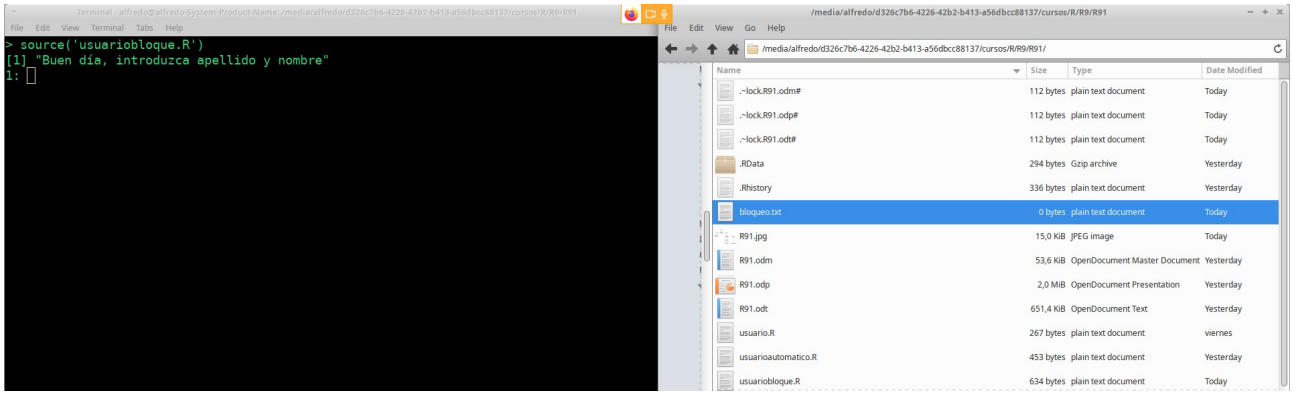


Figura 11

Completa sus datos y el script saldrá automáticamente luego de 3 segundos, y como puede observar al salir de R, el archivo bloqueo.txt se borró permitiendo el ingreso de otro usuario, Figura 12.

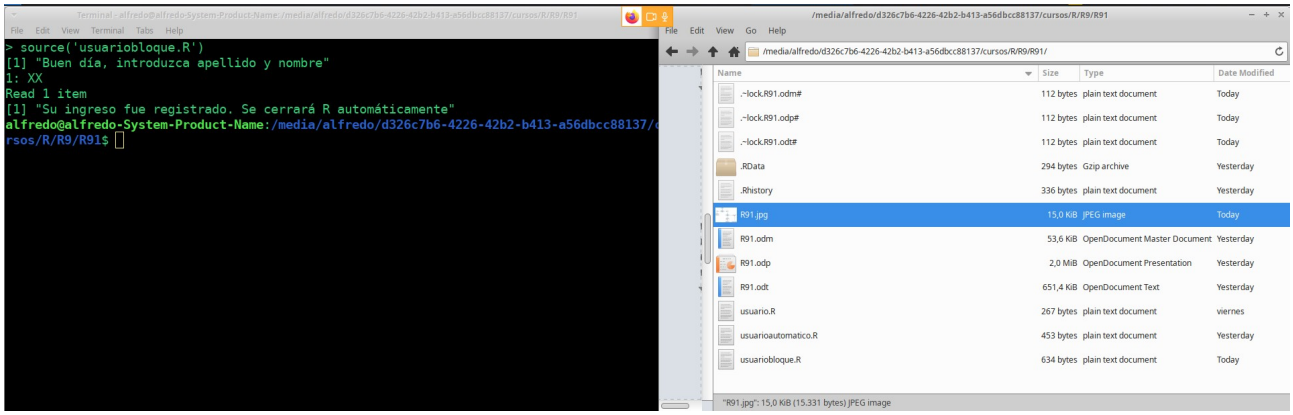


Figura 12

Veamos ahora el ingreso de una persona (usuario1) y que ocurre si mientras está trabajando ingresa otra (usuario2).
El usuario está por ejecutar el script usuariobloqueo.R, por lo que aun no se ha creado bloqueo.txt, Figura 13

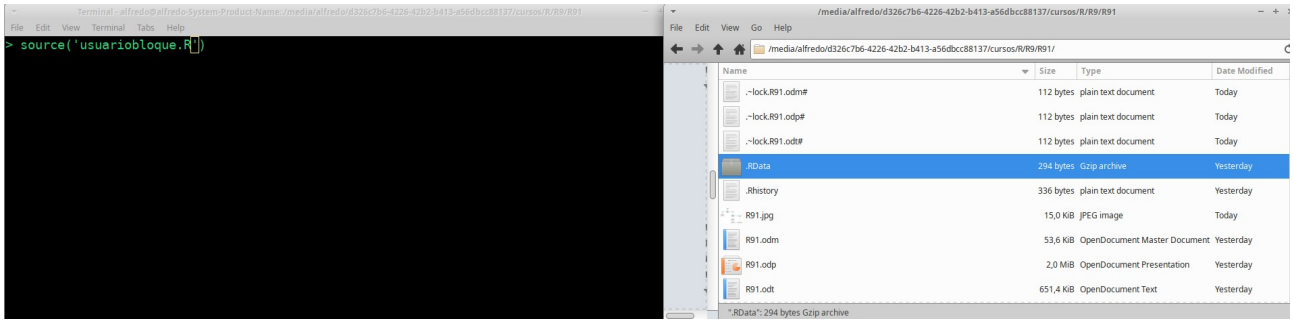


Figura 13

el usuario ingresa al script y por ende aparece bloqueo.txt en el directorio, Figura 14

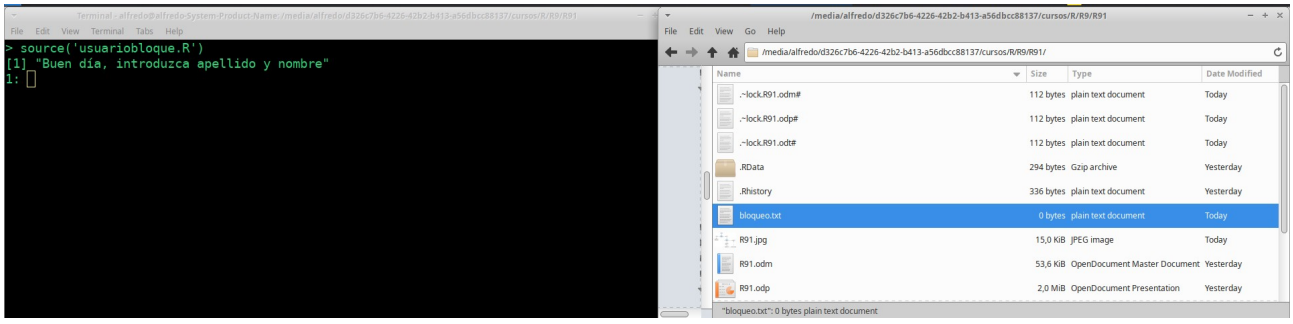


Figura 14
Mientras el usuario1 está en el script, intenta desde otra terminal de la red ingresar el usuario2, que se visualiza en la consola de abajo. Como podemos ver intentó ejecutar usuariobloqueo.R, pero al hallarse bloqueo.txt en el directorio no le permitió el ingreso y salió de R, Figura 15.

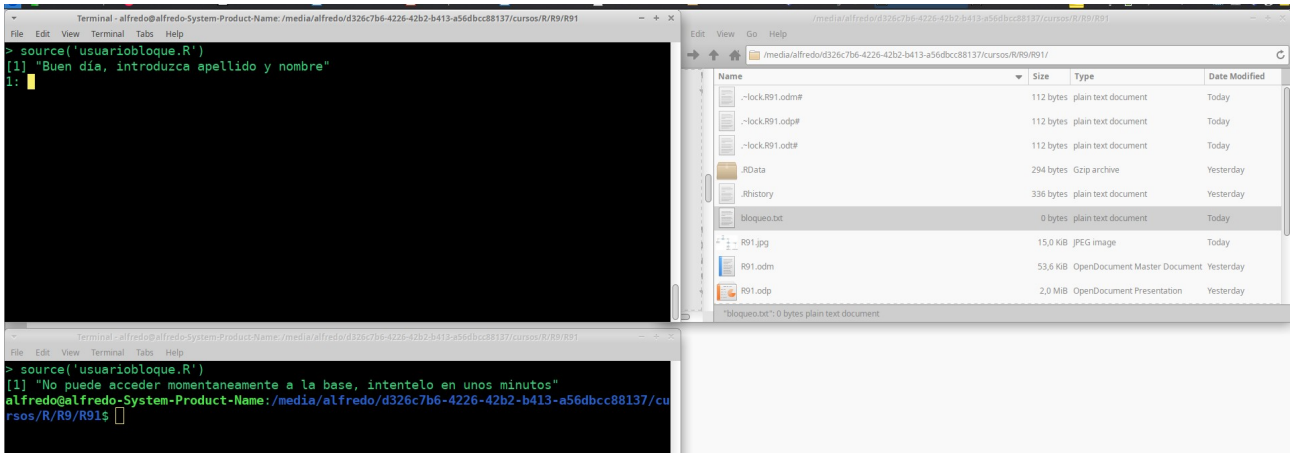


Figura 15

Solución 3

Podría ocurrir que no hubiera un script para ejecutar, sino que simplemente un usuario ingresa a un espacio de trabajo y carga datos en una base de datos o realiza análisis de los mismos. Es claro que la concurrencia a la misma base traerá los mismos problemas que se mostraron en el sencillo ejemplo anterior. En este caso se podrían utilizar las funciones .First() y .Last() que se ejecutan automáticamente al ingresar y salir del espacio de trabajo, respectivamente. Este tema ha sido tratado en el módulo 1 de este curso.

Para lograr nuestro objetivo, evitando la concurrencia, colocamos en la función .First() el chequeo de existencia de bloqueo.txt y su creación en caso de no existir, mientras que en la función .Last() pondremos el borrado de bloqueo.txt.

La función .First() quedaría así

```
.First<-function(){

options(warn=-1)

if(file.exists('bloqueo.txt')==TRUE){

print('No puede acceder momentaneamente a la base, intentelo en unos minutos')

Sys.sleep(3)

quit(save = 'no', status = 0, runLast = FALSE)

}else{

file.create('bloqueo.txt')

}

options(warn=0)

}
```

copie el código anterior y péguelo en la consola. Con eso construirá la función .First() que solo se inicia al ingresar al espacio de trabajo.

La siguiente es la función .Last(), que tendrá el siguiente código

```
.Last<-function(){
```

```
file.remove('bloqueo.txt')

}
```

Copie el código de la función y péguela en la consola.
Veamos ahora el funcionamiento. En la figura siguiente vemos un usuario presto a ingresar a R en el espacio de trabajo cuyo directorio se muestra a la derecha. No existe aun bloqueo.txt en el directorio, Figura 16.

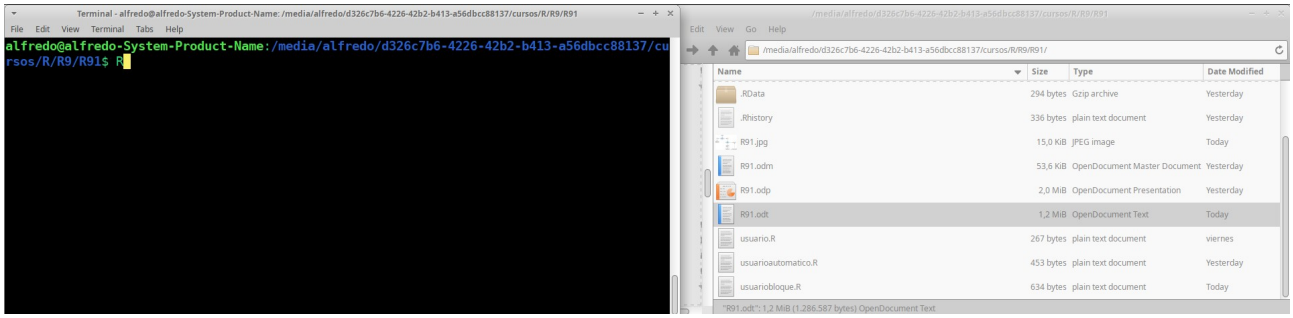


Figura 16

El usuario oprime enter, ingresando al espacio de trabajo, como podemos ver se creo el archivo bloqueo.txt, Figura 17. Nadie podrá ingresar al espacio de trabajo mientras el usuario no salga del mismo.

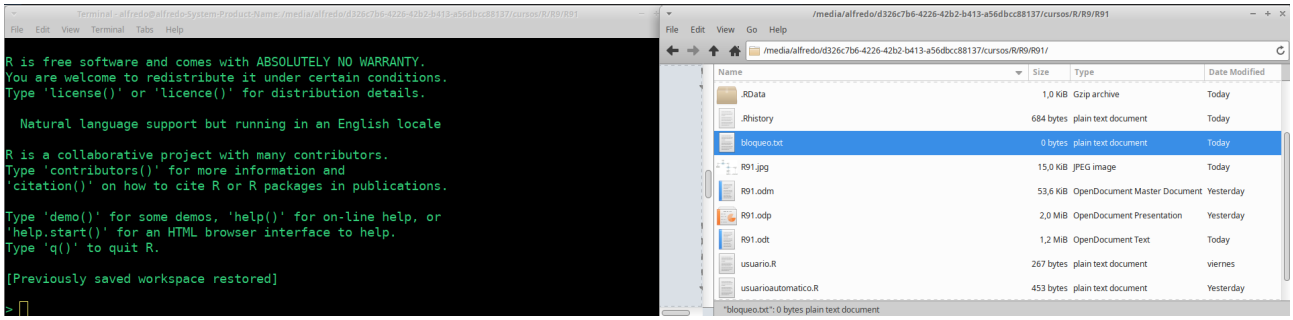


Figura 17
Si mientras el usuario está en el espacio de trabajo, vemos que otro no lo puede hacer. El usuario está utilizando la consola de abajo de la Figura 18.

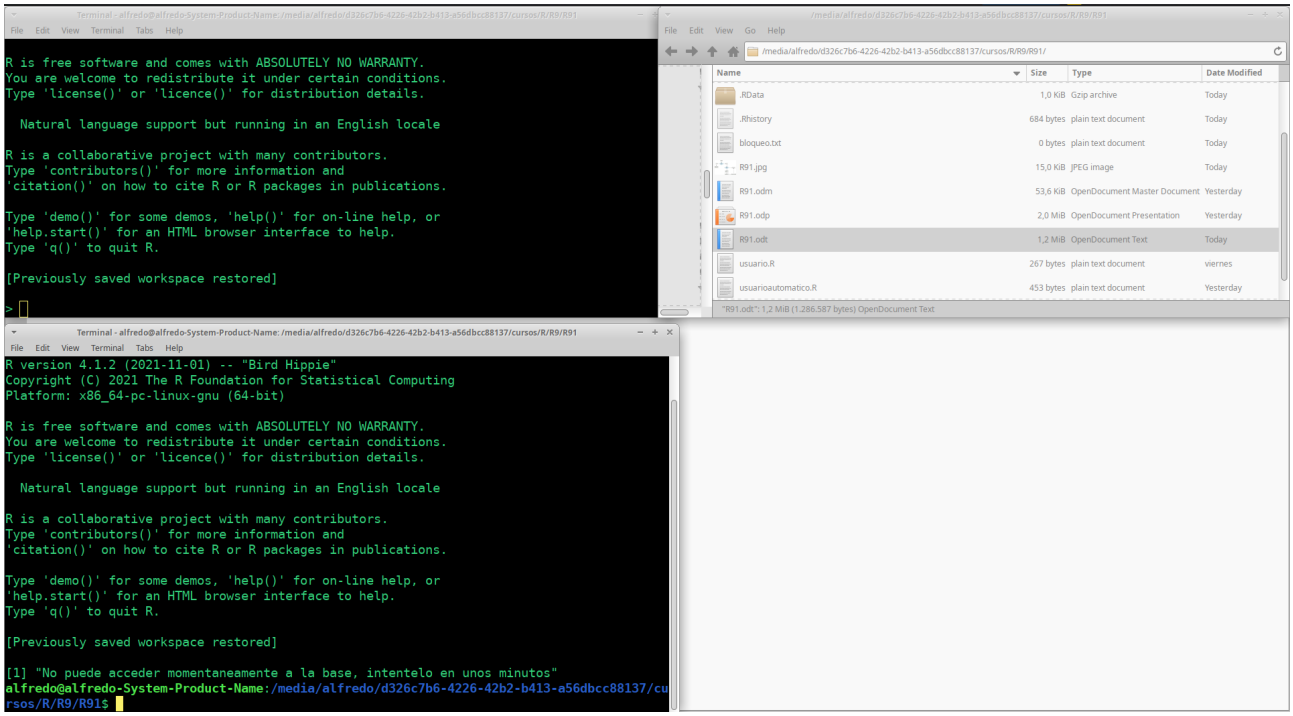


Figura 18

Módulo 9 – clase 2

Vinculación espacio de trabajo con pendrive

Lo habitual cuando trabajamos con R es que nos mantengamos en un espacio de trabajo. Allí creamos nuestros objetos (data.frames, vectores, matrices, listas, variables, funciones y salidas de nuestros análisis de datos). Estos quedan almacenados en el archivo normalmente con la denominación .RData. Son pocos los usuarios que hacen un backup de esta información, en cierta medida ayudados por R, porque la verdad es que rara vez se corrompe un espacio de trabajo.
En esta y en la próxima clase veremos como hacer un backup del espacio de trabajo o de data.frames individuales. Este tema ha sido introducido en módulos anteriores pero en estas clases veremos aplicaciones avanzadas. En este módulo haremos una optimización del tema, dando a R la posibilidad de que este backup lo haga automáticamente a intervalos de tiempo preestablecido.
En primer lugar recordemos algunas funciones útiles para este fin
Funciones útiles

Función getwd()

Esta función permite conocer el camino de nuestro espacio de trabajo. Se ejecuta con
> getwd()
genera una salida como se muestra en la Figura 1

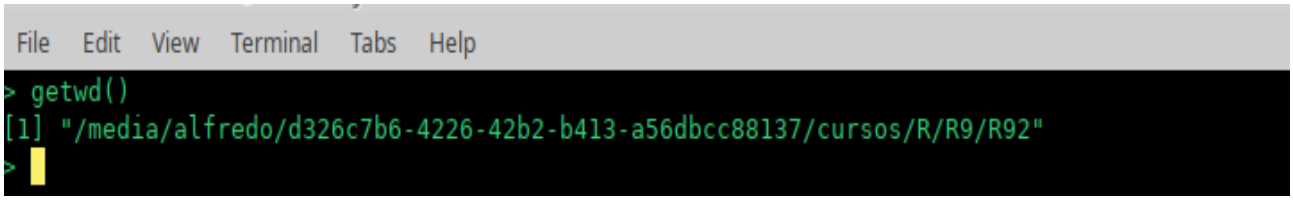


Figura 1

La Figura 1 muestra una estructura de directorios clásica de sistema operativo Linux, pero las diferencias son menores con otros sistemas operativos. Como vemos en este caso el espacio de trabajo está en una carpeta de nombre R92, que corresponde a la clase que estamos dictando. Podemos ver que el primer directorio es media, dentro de él existe el directorio alfredo y allí se monta un disco rígido cuyo nombre es identificado con una larga sucesión alfanumérica.

Si vamos a ese directorio con una administrador de archivos, podemos ver que además de unos archivos de texto tenemos los dos archivos del espacio de trabajo .RData y .RHistory, Figura 2.

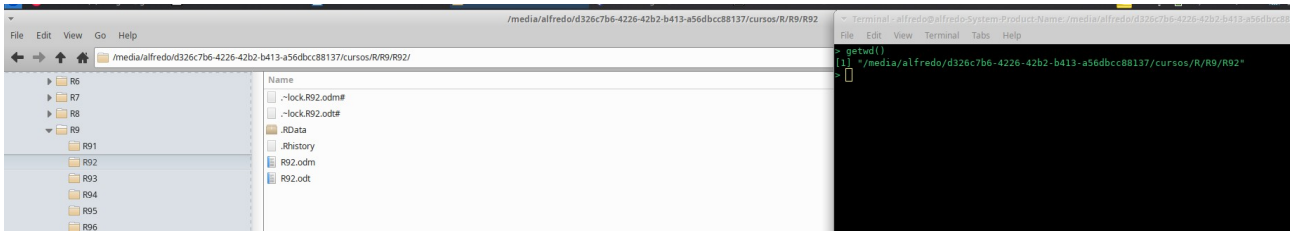


Figura 2

Estos archivos habitualmente se hallan ocultos, razón por la cual pueden no aparecer. Pero, si getwd() le indicó un camino al directorio, allí deben estar.

Un buen complemento de la función getwd() es la función dir(). Ésta nos indica desde la consola los archivos presente en el directorio, sin necesidad de análisis desde un adminstrador de archivos. Si ejecuta

```
> dir()
```

mostrará los archivos, excluyendo los archivos ocultos y transitorios. En cambio si ejecuta

```
> dir(all.files=TRUE)
```

mostrará todos los archivos, Figura 3.

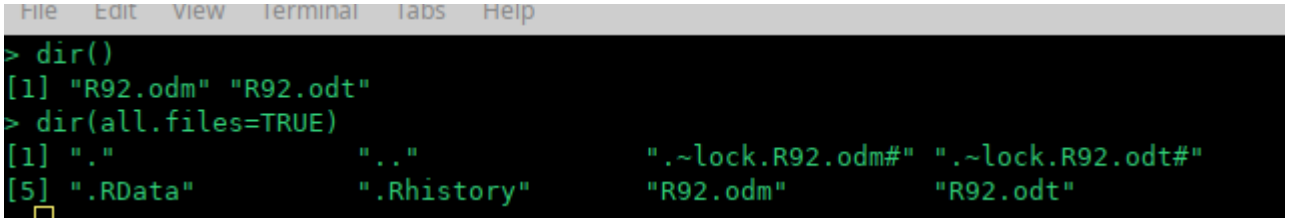


Figura 3

Nuestro próximo objetivo ser guardar el archivo .RData, pero primero recordemos otra función que incorporamos en el módulo 1.

Función setwd()

Esta función nos permite cambiar de directorio. La forma general de uso es

```
> setwd("camino del directorio que deseamos ingresar")
```

En la Figura 4, se muestra un cambio de directorio, dejando el R92, para pasar al R9. Podemos comprobar este cambio con la funcion getwd()

```
> getwd()
[1] "/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9/R92"
> setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9")
> getwd()
[1] "/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9"
```

Figura 4

Nuestro espacio de trabajo aun no tiene objetos por lo que crearemos uno muy sencillo, una variable a que almacene el número 1.

```
> a<-1
```

compruebe que la misma se halla en el espacio de trabajo

```
> a
```

[1] 1

Identificación del pendrive

A través de un administrador de archivos hay que investigar el sitio de montaje del pendrive para poder trazar el camino. En algunos sistemas operativos el disco rígido se llama C y los pendrive ingresan con otra letra. En Linux, el pendrive se monta en el directorio system/media/alfredo. Obviamente el último elemento (alfredo) cambiará en otras computadoras. En la Figura 5, se muestra un disco duro de almacenamiento independiente del disco primario que se identifica con una larga cadena de caracteres y el pendrive, identificado por el Label: B1DD-1E4D. Como podemos ver tiene varias carpetas en su interior. Además tenemos la ruta en la parte superior

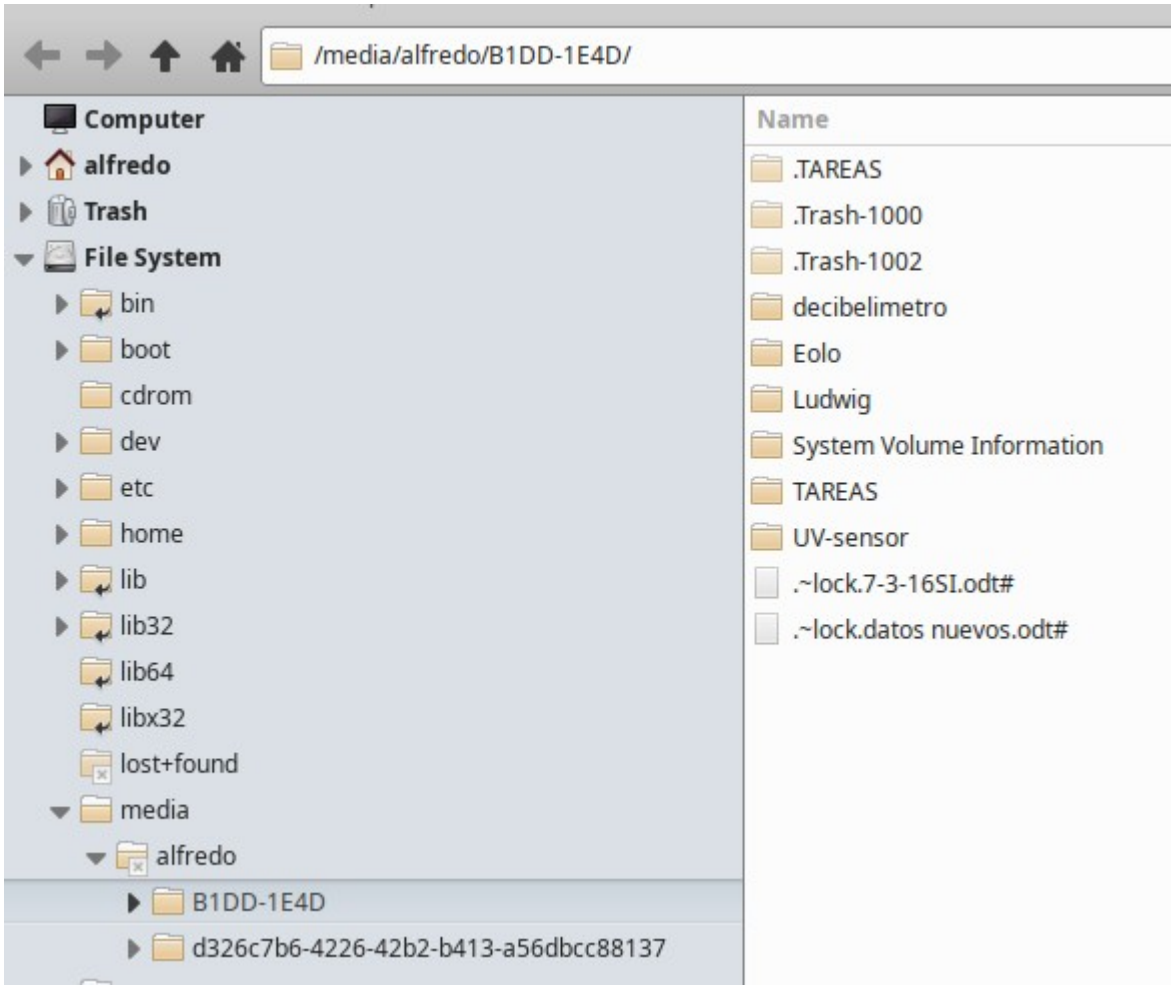


Figura 5

Si deseamos cambiar el directorio de nuestro espacio de trabajo al pendrive, solo tenemos que utilizar la función setwd() con el camino al pendrive o a algunas de sus carpetas si lo desearamos. En primer lugar cambiaremos de espacio trabajo, ingresando al pendrive, para ello utilizamos

```
> setwd("/media/alfredo/B1DD-1E4D/")
```

podemos comprobar que estamos allí con

```
> getwd()
```

```
[1] "/media/alfredo/B1DD-1E4D"
```

y convencernos viendo su contenido

```
> dir(all.files=TRUE)
```

```
 [1] "."                          ".."
 [3] ".~lock.7-3-16SI.odt#"       ".~lock.datos nuevos.odt#"
 [5] ".TAREAS"                    ".Trash-1000"
 [7] ".Trash-1002"                "decibelimetro"
 [9] "Eolo"                       "Ludwig"
[11] "System Volume Information" "TAREAS"
[13] "UV-sensor"
```

Creando backup del espacio de trabajo

Como mencionamos anteriormente, todo el espacio de trabajo se halla en el archivo .RData. Para guardar el espacio de trabajo utilizamos la función save.image(), indicándole qué y con nombre.

Supongamos que deseamos hacer un backup diario en el pendrive, entonces quizás nos conviene colocarle un nombre identificatorio. Le daremos el nombre R29- + fecha + .RData, de manera que cada día se cree un archivo cuyo nombre quedará algo así: R29-2023-03-08.RData, esto nos indica que se ha guardado la información del espacio de trabajo abierto en el directorio R29, el día 8/3/2023. Para ello utilizamos el código

```
> save.image(file=paste("R92",Sys.Date(),".RData",sep=""))
```

como podemos ver para formar el nombre del archivo utilizamos la función paste() que es útil para concatenar cadenas de caracteres. Puede revisar estos conceptos en el módulo 1.
Podemos ver el resultado del proceso, con la función dir(). Observamos en el pendrive un archivo con el nombre R29-2023-03-08.RData como elemento 11 del listado, Figura 6.

```
> save.image(file=paste("R92-",Sys.Date(),".RData",sep=""))
> dir(all.files=TRUE)
 [1] "."                          ".."
 [3] ".~lock.7-3-16SI.odt#"       ".~lock.datos nuevos.odt#"
 [5] ".TAREAS"                    ".Trash-1000"
 [7] ".Trash-1002"                "decibelimetro"
 [9] "Eolo"                       "Ludwig"
[11] "R92-2023-03-08.RData"       "System Volume Information"
[13] "TAREAS"                     "UV-sensor"
```

Figura 6

El archivo R29-2023-03-08.RData, es un archivo muy pequeño. Un espacio de trabajo con años de ingreso de datos, análisis y variables, rara vez superará 1 MByte. Por lo que la cantidad de archivos que se pueden guardar en un pendrive de 16GB, puede ascender fácilmente a varias decenas de miles de días. Teniendo en cuenta que un humano puede vivir alrededor de 30000 días, podrá almacenar en un pendrive de esas dimensiones prácticamente toda la vida de un espacio de trabajo.
Así en su pendrive quedarán archivos identificados con el directorio y/o espacio de trabajo y la fecha. Si ocurriera una catástrofe en su espacio de trabajo, que seguramente será por la mala acción de algún usuario, antes que por una falla de R, podrá retroceder en el tiempo buscando el archivo que requiera para restablecer la información.
Una vez que haya realizado el backup, vuelva al espacio de trabajo nuevamente con setwd()

```
> setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9/R92")
```

comprobamos que estamos nuevamente en el espacio deseado

```
> getwd()
```

y comprobamos si los objetos se hallan en él con la función ls(). En este caso solo creamos uno que se llama a

```
> ls()

[1] "a"
```

vemos que está el valioso objeto. Supongamos que algún humano hizo una macana, corrompiendo el espacio de trabajo, que no necesariamente signifique que no se pueda abrir. Simplemente puede haber sido el borrado de objetos valiosos. Simule ser usted quien lo hace.
Borre el objeto a

```
> rm(a)
```

comprobamos que se ha borrado

```
> ls()
```

character(0)
Si cierra la sesión guardando, el objeto **a** estará perdido.

Pero su valioso pendrive lo salvará
Copie el archivo del pendrive al directorio donde se halla el espacio de trabajo, como se muestra en la Figura 7

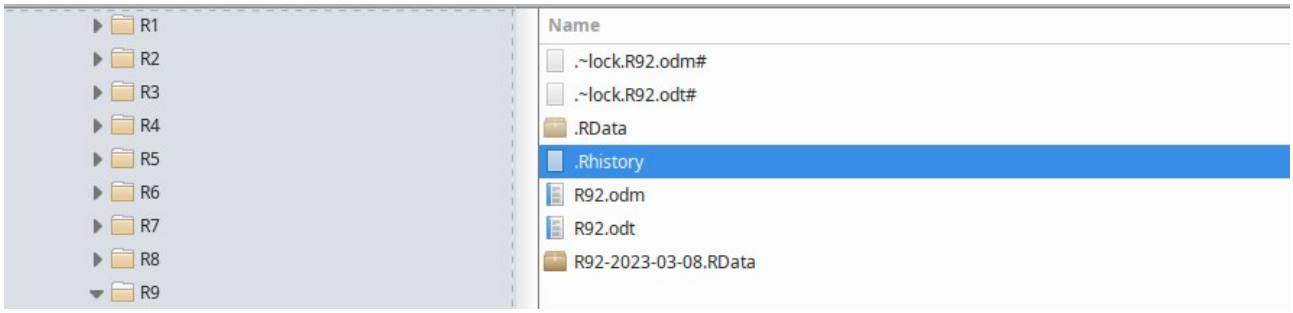


Figura 7

podemos ver el archivo R92-2023-03-08.RData
Asegurese de estar en el espacio de trabajo
En la consola ejecute

```
> load("R92-2023-03-08.RData")
```

compruebe si está su querido objeto "a"

```
> ls()
```

[1] "a"
El backup ha dado sus frutos. Hemos recuperado el espacio de trabajo, con sus objetos.
En la próxima clase veremos como hacer que su espacio de trabajo haga un backup automático del día anterior, antes de comenzar una nueva sesión con R.

Módulo 9 – clase 3

Vinculación automática del espacio de trabajo con pendrive

Ya hemos visto lo sencillo que es realizar un backup del espacio de trabajo y más valioso aun será cuando usemos el mecanismo de rescate de nuestros datos y lo logremos con éxito. En esta clase veremos un mecanismo automática para hacer el backup, que plantearemos que se realice de manera automática, con mínima intervención de nuestra parte.
Imaginemos una situación. Soy usuario de un espacio de trabajo que diariamente realizo ingreso de datos, análisis y genero resultados. Cuando dejo el espacio de trabajo, prolijamente salgo del mismo por lo que no debería tener ningún percance con los datos. La idea es incorporar al espacio de trabajo un mecanismo que diariamente cuando ingreso por primera vez, antes de comenzar a trabajar y modificar el espacio de trabajo se haga un backup, que sea almacenado en una unidad externa como un pendrive. Este backup llevará un identificador del espacio de trabajo y la fecha. Así al finalizar el año tendremos no más de 365 archivos de tipo .RData con los datos de cada día.
Si tiene algún percance, como por ejemplo le robaron la computadora o tuvo un severo daño en los medios de almacenamiento, podrá recuperarla. Nunca deje el pendrive junto con la computadora. Es probable que el ladrón se lleve las dos cosas. En las clases 4 y 5 veremos como podemos cuidar nuestra información con R, aun contra los malechores!!

Para comenzar veremos algunas rutinas necesarias para cumplir con nuestro objetivo
Rutinas

Creación de vector de fechas

A continuación veremos un mecanismo para crear un objeto de tipo vector que almacene la fecha, de manera que cuando abrimos nuestro espacio de trabajo guarde la fecha actual (año-mes-dia). La condición que le pondremos que guarde la fecha una sola vez por día. Es decir que si ingresamos varias veces en el día el vector solo la almacene la primera vez.
En primer lugar creamos en el espacio de trabajo un vector que podemos llamar: vectorfecha y asignaremos como primer elemento la fecha actual con la función Sys.Date(), a la que transformaremos en caracter. Recuerde que Sys.Date() nos da la fecha en formato año-mes-día. Compruébelo

```
> Sys.Date()
```

[1] "2023-03-10"
Entonces creamos el vector, con la primer fecha

```
> vectorfecha<-c(as.character(Sys.Date()))
```

Si comprobamos su contenido

```
> vectorfecha
```

[1] "2023-03-10"
Lo que deseamos ahora es que la primera vez que en el día se habra el espacio de trabajo, este vector se actualice con la fecha actual. Seguramente podrá imaginarse decenas de forma de hacerlo. Mostraremos una que nos ha funcionado muy bien.
Tendremos una sentencia if-else que evaluará la última fecha del vectorfecha, si no coincide con la fecha actual dada por Sys.Date(), colocará en la posición final del vector la fecha actual. En cambio, si la última fecha coincide con la fecha actual, no hará nada. La rutina es

codigo	Explicacion
if(vectorfecha[length(vectorfecha)] !=Sys.Date()){ vectorfecha<-c(vectorfecha, as.character(Sys.Date()))}	compara si el último elemento del vectorfecha es diferente de la fecha actual Si se cumple condición, agrega a vector fecha, la fecha actual

Este código puede colocarlo en la función .First() de su espacio de trabajo

```
.First<-function()  {

if(vectorfecha[length(vectorfecha)] !=Sys.Date()){

vectorfecha<-c(vectorfecha, as.character(Sys.Date()))}

}
```

al introducir esta función en el espacio de trabajo quedará como muestra la Figura 1

```
> .First<-function()  {
+ if(vectorfecha[length(vectorfecha)] !=Sys.Date()){
+ vectorfecha<-c(vectorfecha, as.character(Sys.Date()))}
+ }
>
```

Figura 1

podemos comprobar el contenido del vectofecha

```
> vectorfecha
```

[1] "2023-03-10"

Bien salga del espacio de trabajo, por supuesto guardando los cambios de manera que el vector y la función .First permanezcan. Ingrese inmediatamente y repita salidas y entradas las veces que desee. Vea el contenido de vectorfecha

```
> vectorfecha
```

[1] "2023-03-10"

Como puede comprobar, no se ha agregado fecha al vector, ya que estamos en la misma fecha. Mañana ingrese a este espacio de trabajo y verifique el contenido de vector fecha. Seguramente hallará

```
> vectorfecha
```

[1] "2023-03-10" "2023-03-11"

Entonces ya tenemos el primer elemento de nuestro backup automático.

Backup automático

Desarrollaremos un código que nos permita hacer el backup de manera automática con las siguientes características

1- lo haga solo una vez por día. Esta acción deseamos que sea la primera vez del día que se ingresa

2- haga un backup del espacio de trabajo en un pendrive

3- haga un backup en una unidad física de almacenamiento de nuestra PC

4- Nos avise que ha realizado el backup

Estas acciones pueden realizarse desde la ejecución de un script, pero nosotros lo haremos con la función .First().

Comenzaremos por borrar la función .First() que hemos creado, para hacer una nueva

> rm(.First)

comprobamos que la función se ha eliminado. Si no la borra, puede dar error al intentar sobreescribirla.

> .First

NULL

Por comodidad para mostrar los resultados de nuestro código guardaremos la información en el pendrive cuyo label es B1DD-1E4D y en otro directorio que hemos creado en una unidad física de almacenamiento de la PC en este caso el disco rígido de label: d326c7b6-4226-42b2-b413-a56dbcc88137.

Allí hemos creado un directorio cuyo nombre es: AABackupR93. El caprichoso nombre obedece a que deseamos que quede primero en la lista y podamos ver los backup correspondientes, en una misma foto, como lo muestra la .

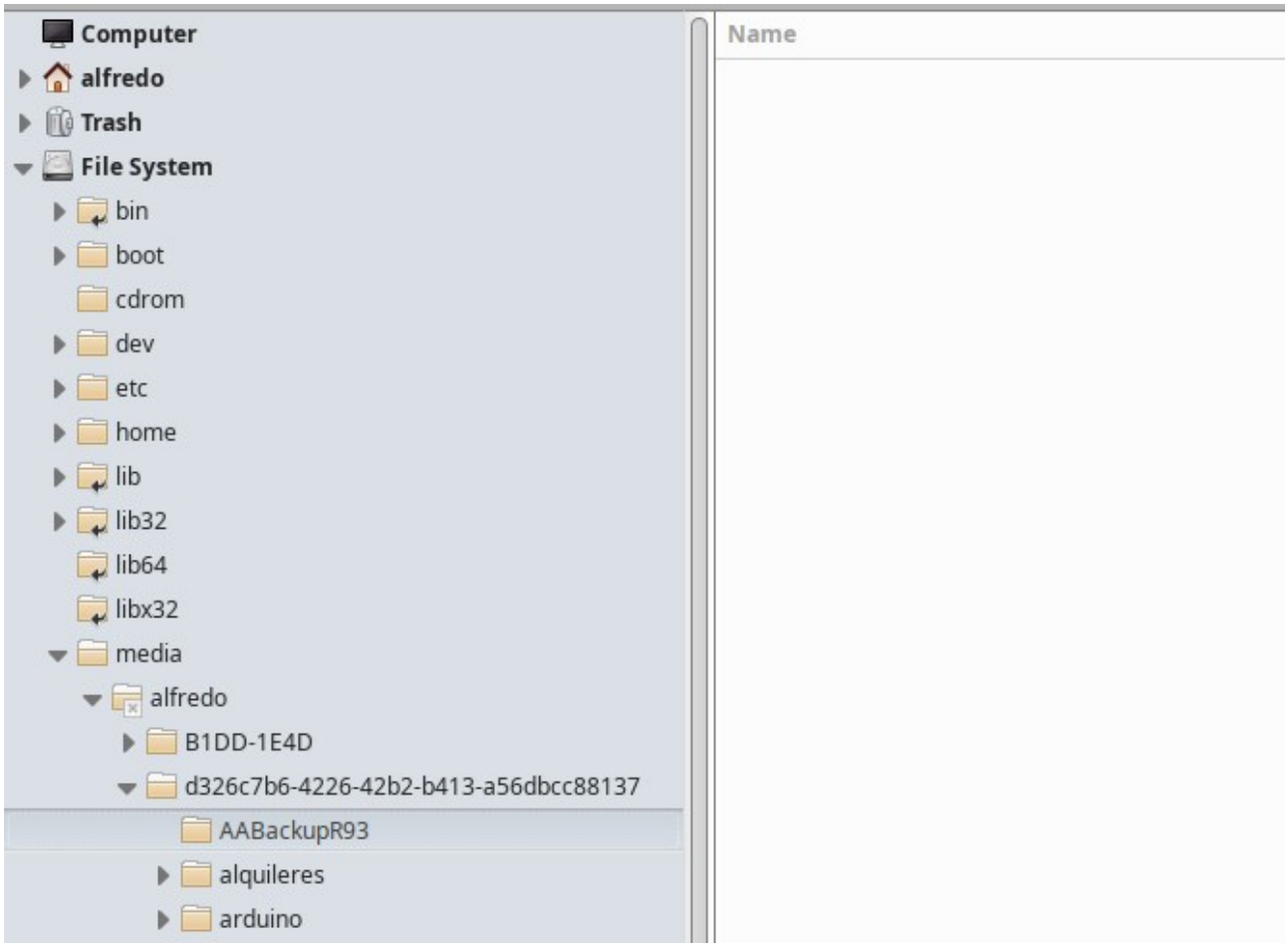


Figura 2. En recuadros rojos puede ver el pendrive: B1DD-1E4D y el directorio AABackupR93

¿Qué debe hacer la función .First()?

1- al ingresar al espacio de trabajo debe chequear si el último elemento del objeto: vectorfecha es diferente de la salida que da Sys.Date()
2- si no es diferente que no haga nada
3- si es diferente debe ejecutar varias acciones, que enumeramos
4- cambiar con la función setwd() al pendrive, cosa que ya sabemos
5- hacer un backup allí dando el nombre del espacio de trabajo y la fecha
6- cambiar al directorio AABackupR93
7- hacer un backup allí dando el nombre del espacio de trabajo y la fecha. Para evitar errores que le de exactamente el mismo nombre.
8- cambiar al directorio donde se halla el espacio de trabajo
9- Que nos avise que se hicieron los backups correspondientes
10- Que nos desee un buen día de trabajo

El código de la función .First() quedaria como se muestra a continuación

código	explicación
.First<-function(){	declara la función
if(vectorfecha[length(vectorfecha)]!=Sys.Date()){	compara si el último elemento de vector fecha es distinto de la fecha actual
readline('Coloque el pendrive para el backup, luego oprima enter')	Nos pide que coloquemos el pendrive y oprimamos enter
setwd("/media/alfredo/B1DD-1E4D/")	cambia de directorio, dirigiendose al pendrive
save.image(file=paste("R93-",Sys.Date(),".RData",sep=""))	guarda la información del espacio R93 con el nombre R93-2023-03-11.RData
setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/AABackupR93/")	Cambia de directorio yendo a AABackupR93
save.image(file=paste("R93-",Sys.Date(),".RData",sep=""))	guarla la información con el mismo nombre
setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9/R93/")	Vuelve al directorio R93, donde tenemos el espacio de

print('Se realizaron los backup correspondiente al día') readline('Que tenga un magnífico día. Oprima enter para continuar') vectorfecha<-c(vectorfecha,as.character(Sys.Date())) } }	trabajo Nos avisa de los backup realizados Nos desea un buen día Actualiza el vectorfecha con la fecha actual Ciera la estructura if Cierra la función .First

Borre la función .First que había creado
> rm(.First)
> .First
NULL

Copie la celda que contiene la función .First() y péguela en su espacio de trabajo, oprima enter

verifique que se halla presente el objeto vectorfecha creado
> vectorfecha
[1] "2023-03-10"

salga del espacio de trabajo guardando la información y reingrese. Si lo está haciendo el mismo día que figura en el último lugar de vector fecha, no deberia ocurrir ninguna acción. Pruebe!
Si todo fue bien realizado cargó su espacio de trabajo y en él, por ahora solo existe vectorfecha con un elemento
> vectorfecha
[1] "2023-03-10"

Engañaremos al sistema para ponerlo a prueba. Cargaremos en vector fecha la fecha del día anterior, para este ejemplo sería 2023-03-09,
>vectorfecha[1]<-"2023-03-09"
> vectorfecha
[1] "2023-03-09"
Bien. Salimos del espacio de trabajo guardando los cambios. Nuestro espacio de trabajo cree que es "Ayer". Reingrese al espacio de trabajo. Ahora .First actuará ya que Sys.Date() nos dará la fecha de hoy y vectorfecha tiene en su última posición la fecha de ayer.
Ni bien ingrese recibirá el texto habitual con versión de R y otros textos, pero se detendrá en
Coloque el pendrive para el backup, luego oprima enter
Si sigue las instrucciones al pie de la letra, es decir coloca el pendrive (No cualquiera, siempre tiene que utilizar el que tiene el label que se halla en el código, en este caso es B1DD-1E4D) y luego oprime enter, recibirá casi instantáneamente los mensajes
[1] "Se realizaron los backup correspondiente al día"
Que tenga un magnífico día. Oprima enter para continuar

Oprima enter y disfrute del día!

Comprobemos si se hizo lo que desamos. En la Figura 3, se muestra el contenido del pendrive donde vemos el archivo R93-2023-03-10.RData

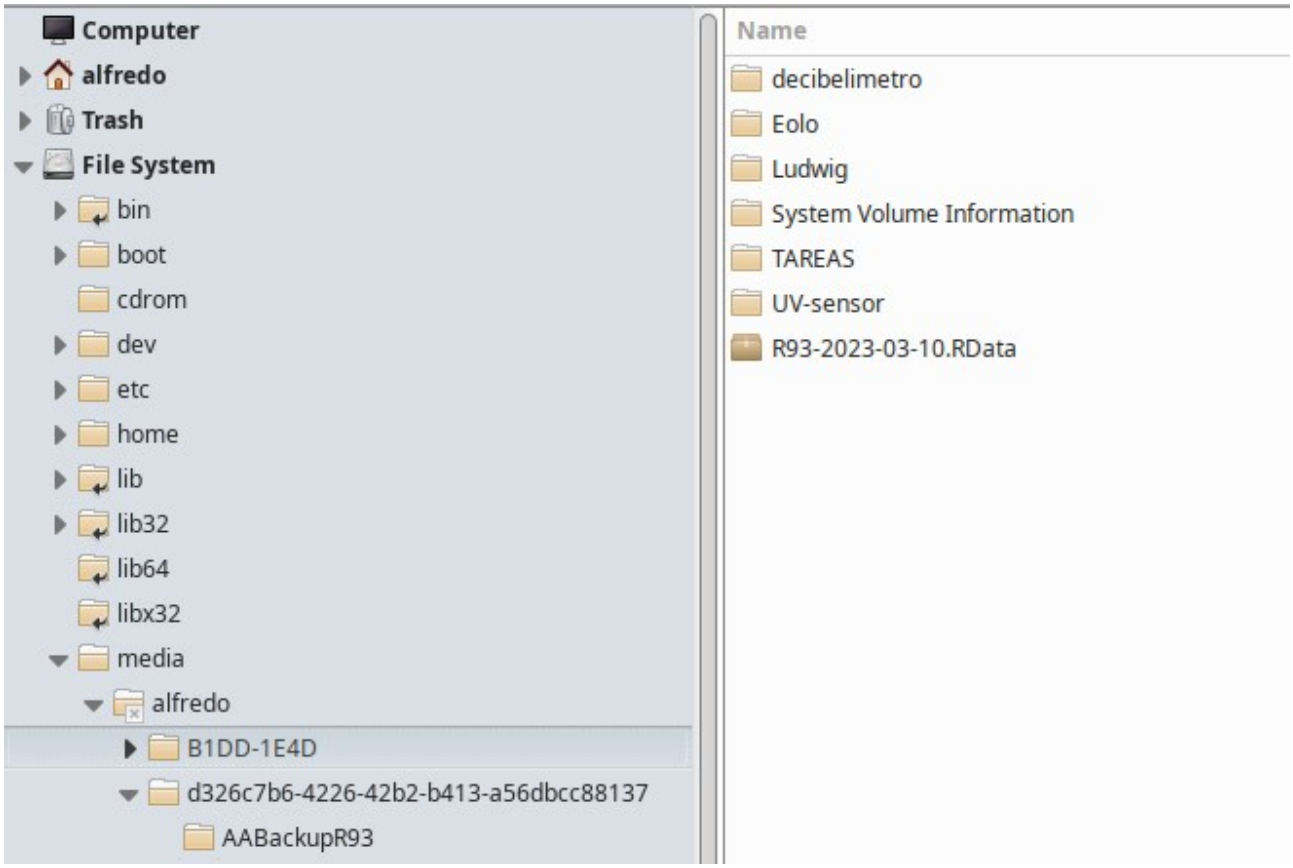


Figura 3

En la Figura 4, vemos el mismo archivo en el directorio AABackupR93

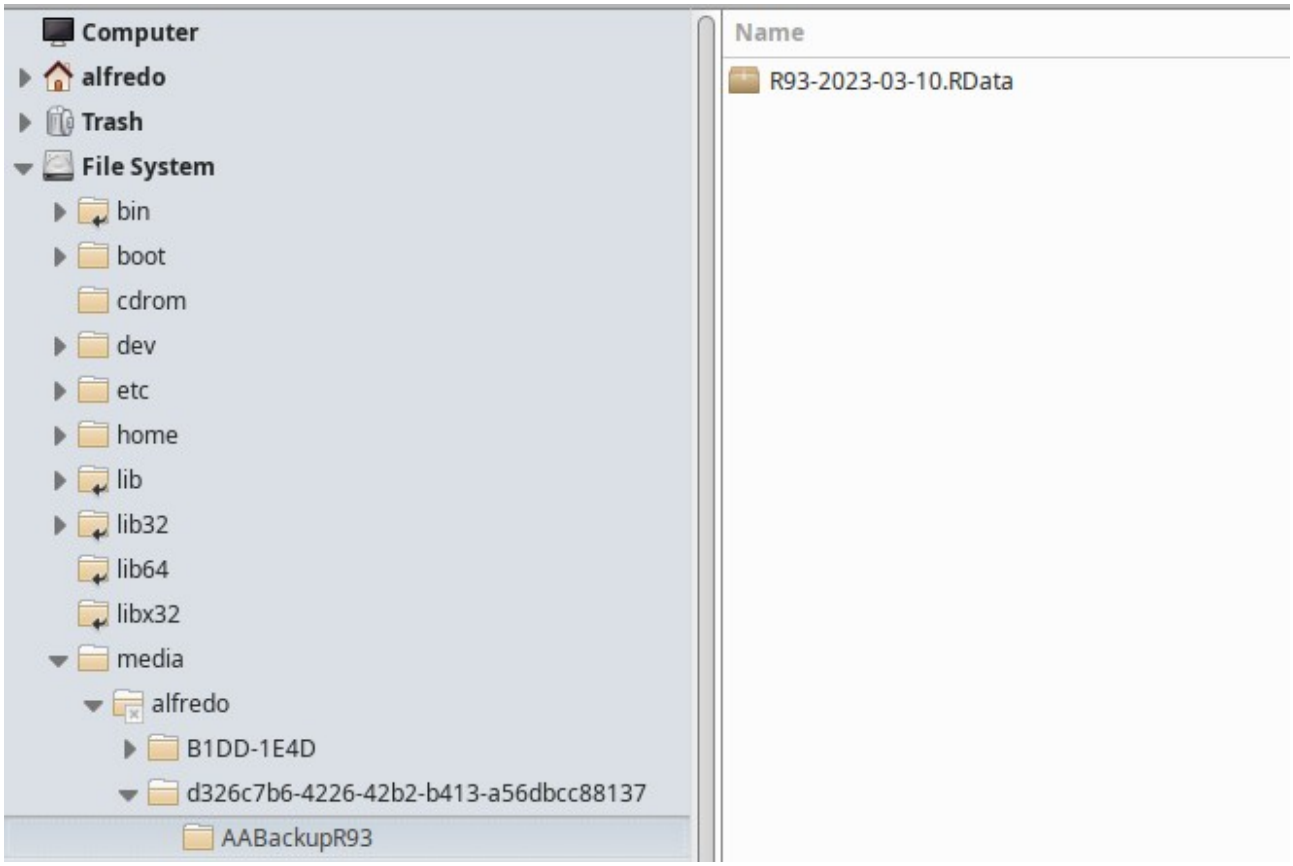


Figura 4

Se supone que luego de un breve descanso, programará la función .First() al menos en sus espacios de trabajo más valiosos.

Otra forma de cumplir el objetivo, con los mismos resultados, es colocar el código desarrollado en un script, que podemos llamar por ejemplo: arranque. Cree una archivo de texto en el mismo directorio con el nombre arranque.R y peque el texto de la tabla, eliminando la fila .First y la última llave

```
if(vectorfecha[length(vectorfecha)]!=Sys.Date()){

readline('Coloque el pendrive para el backup, luego oprima enter')
setwd("/media/alfredo/B1DD-1E4D/")
save.image(file=paste("R93-",Sys.Date(),".RData",sep=""))

setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/AABackupR93/")
save.image(file=paste("R93-",Sys.Date(),".RData",sep=""))
setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9/R93/")

print('Se realizaron los backup correspondiente al día')
readline('Que tenga un magnífico día. Oprima enter para continuar')
vectorfecha<-c(vectorfecha,as.character(Sys.Date()))
}
```

Luego redefina .First de la siguiente manera

```
.First<- function(){
source('arranque.R')
}
```

obtendrá el mismo resultado.

Veamos una aplicación completa utilizando el script de la clase R91, con el que registramos la asistencia. En la tabla siguiente recuperamos el script

usuario.R	explicaciones
print('Buen día, introduzca apellido y nombre') nombre<-as.character(scan(file = "", what = "",nmax=1,sep="\n",nlines=1)) filas<-nrow(usuariollegada) usuariollegada[filas+1,1]<-nombre usuariollegada[filas+1,2]<-as.character(date()) print('Su ingreso fue registrado')	Muestra un mensaje en pantalla solicitando el nombre Una vez introducido el nombre lo asigna a la variable nombre cuenta las filas del data.frame usuariollegada en la fila siguiente columna 1 coloca el nombre en la misma fila pero columna 2 coloca la fecha le informa al usuario que los datos fueron registrados.

Lo modificaremos de la siguiente manera, hallando en rojo la porción introducida. Por supuesto este script necesitará que cree el data.frame usuariollegada, o bien lo recupere desde el espacio R19

arranque.R	explicaciones
if(vectorfecha[length(vectorfecha)]!=Sys.Date()){ readline('Coloque el pendrive para el backup, luego oprima enter') setwd("/media/alfredo/B1DD-1E4D/") save.image(file=paste("R93-",Sys.Date(),".RData",sep="")) setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/AABackupR93/")	Compara vector fecha con fecha actual. Si se cumple la desigualdad Pide el pendrive cambia directorio al pendrive guarda en el pendrive un backup del espacio de trabajo cambia a un directorio de una unidad física de nuestra PC

save.image(file=paste("R93-",Sys.Date(),".RData",sep=""))	guarda el espacio de trabajo
setwd("/media/alfredo/d326c7b6-4226-42b2-b413-a56dbcc88137/cursos/R/R9/R93/")	regresa al espacio de trabajo
print('Se realizaron los backup correspondiente al día')	Nos informa que realizó los backup preestablecidos
readline('Que tenga un magnífico día. Oprima enter para continuar')	Nos despide
vectorfecha<-c(vectorfecha,as.character(Sys.Date()))	Actualiza vectorfecha
}	
print('Buen día, introduzca apellido y nombre')	Muestra un mensaje en pantalla solicitando el nombre
nombre<-as.character(scan(file = "", what = "",nmax=1,sep="\n",nlines=1))	Una vez introducido el nombre lo asigna a la variable nombre
filas<-nrow(usuariollegada)	cuenta las filas del data.frame usuariollegada
usuariollegada[filas+1,1]<-nombre	en la fila siguiente columna 1 coloca el nombre
usuariollegada[filas+1,2]<-as.character(date())	en la misma fila pero columna 2 coloca la fecha
print('Su ingreso fue registrado')	le informa al usuario que los datos fueron registrados.

La ventaja de esta última alternativa es que solo hará backup si ejecutamos el script. Puede ser útil en algunas circunastancias y no en otras.
Finalmente, recuerde que no puede introducir cualquier Pendrive, solo funcionará con el pendrive cuyo Label figura en el script. Reserve un pendriva para los backups, será más fácil para buscar un archivo si es que lo necesita.

Módulo 9 – clase 4

Envíos de email con R

Como no podía ser de otra manera R tiene la posibilidad de enviar correos electrónicos. Podríamos preguntarnos: ¿Para que complicarnos si ya existen buenas aplicaciones para el manejo de los correos electrónicos? La respuesta es directa y sencilla. Estas funciones de R nos permitirán enviar emails automáticos desde la ejecución de scripts.
Analicemos una aplicación que se encuentra en marcha en el Centro Universitario de Estudios Medioambientales (CUEM). En el CUEM se realizan diversas mediciones y estudios medioambientales. Uno de los servicio que se brindan a la población es el análisis químico y microbiológico del agua. Este estudio involucra cerca de 30 mediciones, realizadas por diferentes personas con entrenamiento especializado, para cada una de ellas. Estas personas no asisten regularmente al CUEM sino cuando ingresa una muestra de agua a analizar. Un secretario, bien podría enviar un mensaje por algún mecanismo de los varios existentes, avisándo a cada uno de los responsables, que ha ingresado una muestra. El secretario podría olvidarse de avisar a alguno de ellos y se atrasaría el análisis.
En el CUEM el registro de ingreso de las muestras de agua, el ingreso de los datos de cada una de las mediciones, el análisis de estos datos y la construcción de los informes son realizados por un script, llamado Atlantis que se halla en la versión 3.1. Por supuesto que el script cumple otras funciones como indicar cuales muestras se deben medir, si las mediciones pasaron o no el control de calidad, si alguna muestra tiene algún valor de componente tóxico que supere los valores permitidos por la legislación vigente, si se le presentaron problemas en la medición a los encargados, por mencionar algunas.
Por supuesto que una función incorporada es la reemplazar a un posible secretario, en el aviso de ingreso de muestra. Atlantis realiza esto automáticamente a través de email a los responsables de cada determinación. Además como cada responsable dispone de 2 semanas para realizar la medición, si ésta no ha sido realizada diariamente enviará un email a cada responsable que se halle atrasado en su trabajo. Por supuesto también se encargará de descontarle puntos a su puntaje anual, que permite organizar un ranking de eficiencia y cumplimiento de los responsables.
Imagínese otras funciones que podría cumplir un script que envie automáticamente emails. Por ejemplo puede colocar un sensor de temperatura en cada una de sus heladeras y freezers controladas por algun micropocesador como podría ser un Arduino y comunicar a éste con el script. En el

programa del Arduino usted puede fijar un rango de variación aceptable para cada refrigerador y si se violan los límites, por ejemplo por un corte de luz, que el scritp lo detecte y envie uno o varios emails a responsables de actuar en esta situación.
De riendas libres a su imaginación y comience a pensar donde lo va a utilizar.

Bibliotecas

Para poder llevar adelante esta función R necesitará la instalación de dos bibliotecas: mailR y rJava, que puede descargar de los repositorios habituales. Instale las bibliotecas como se ha explicado en modulos previos. Si tuviera dificultades instale bibliotecas acorde a la fecha de su versión de R y consulte los conceptos relacionados a este tema en el módulo 1 de este curso.
Una vez que haya logrado la instalación cargue las bibliotecas en su espacio de trabajo

```
> library(mailR)
> library(rJava)
```

Puede existir problema con rJava, por la versión de Java y sus bibliotecas. Antes de instalar las bibliotecas mencionadas tendrá que trabajar en su sistema operativo.
Si trabaja en Linux, abra una terminal o consola con la combinación de teclas Ctrl+Alt+T
Instale las bibliotecas de Java

```
~$ sudo apt-get install default-jre
~$ sudo apt-get install default-jdk
```

ejecute en la línea de comando

```
~$ sudo R CMD javareconf
```

Con los comandos indicados adaptará las bibliotecas de java para su versión de R.
Reinicie la computadora.
Arranque R de la manera habitual en el directorio deseado, en un nuevo o en uso espacio de trabajo
Instale las bibliotecas necesarias

```
> install.packages("rJava",depedencies=TRUE)
> install.packages("mailR",depedencies=TRUE)
```

cargue las bibliotecas

```
> library(mailR)
> library(rJava)
```

Deberá tener una serie de parámetros de su conexión a internet. Para obtener los parámetros de envío abra su administrador de correo electrónico, haga click derecho sobre la cuenta y elija settings, Figura 1. Explicaremos el procedimiento para el administrador de correo Thunderbird, que de allí podemos obtener los datos. Si fuera otro administrador de correos electrónicos, no habrá mucha diferencia, los parámetros deben estar en algún lado.

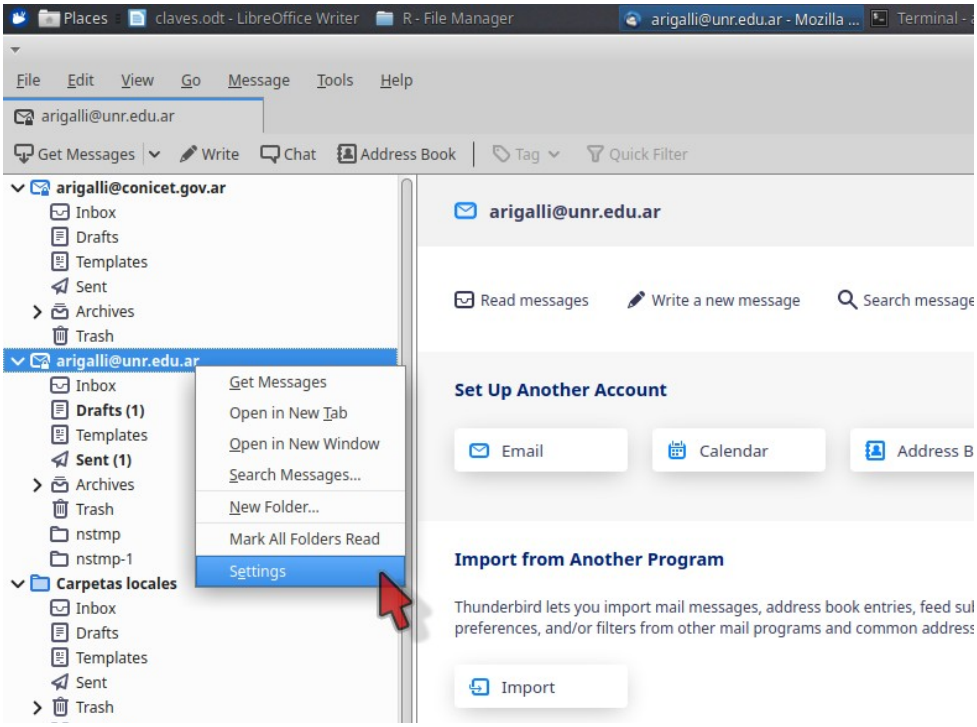


Figura 1

al hacer click en Settings se desplegará una ventana, haga click en Outgoing Server (SMTP), que le mostrará una lista de los correos salientes, Figura 2.

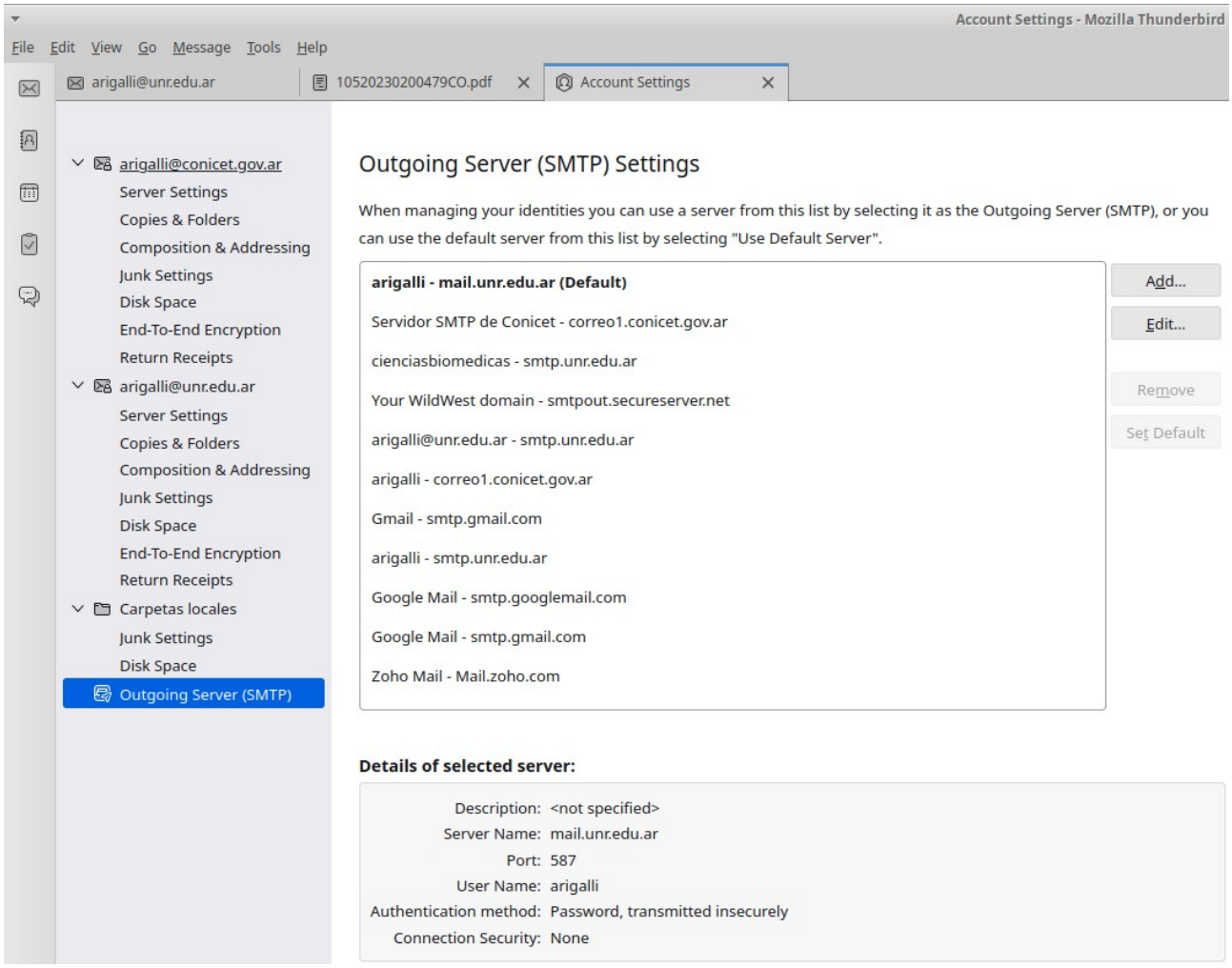


Figura 2

Elija uno de los correos salientes que use habitualmente y conoce que funciona, en este caso se seleccionó arigalli - mail.unr.edu.ar (Default).
Debajo en "Details of selected server:" hallará un listado de parámetros.
De allí tomará los parámetros para el argumento smtp de la función send.mail() que pertenece al paquete mailR. La función send.mail() es la encargada de enviar el mail cuando se ejecuta como cualquier función.

La función send.mail() para el envio de correo electrónico

A continuación mostramos el código en una sola línea teniendo en cuenta que
1- el email se envia desde la dirección: arigalli@unr.edu.ar.
2- el envío se realiza a la misma dirección y a otra dirección: arigalli@conicet.gov.ar
que estoy enviando un email que llegará a la misma dirección desde donde hago el envio y a una cuenta de conicet.
3- En el subject dirá: "mail desde ATLANTIS 3.1"
4- En el cuerpo del mail dirá: ·Este es un mail enviado por ATLANTIS para saludarlo por el día del biotecnólogo"
5- En el argumento smtp irán los parámetros obtenidos de administrador de correo electrónico. mostrado en la Figura 2. Como puede ver, host.name corresponde al dato "Server Name", port figura como Port, user.name figura como User Name. El parámetro passwd, es la contraseña que le han dado o ha fijado usted en el servidor de correo. En este caso se pone con ********, pero

habitualmente es una cadena alfanumérica con restricciones impuestas por cada servidor El parámetro ssl colóquelo en FALSE. El parámetro authenticate colóquelo en TRUE al igual que send.

```
>
send.mail(from="arigalli@unr.edu.ar",to=c("arigalli@unr.edu.ar","arigalli@conicet.gov.ar"),subject
="mail desde ATLANTIS 3.1", body="Este es un mail enviado por ATLANTIS para saludarlo por el
día del biotecnólogo",smtp=list(host.name="mail.unr.edu.ar", port=587, user.name
="arigalli",passwd="********", ssl=FALSE),authenticate=TRUE,send=TRUE)
```

Al finalizar oprima enter y se enviará el mail. Debe aparecer un mensaje

```
[1] "Java-Object{org.apache.commons.mail.SimpleEmail@52a86356}"
```

Analicemos nuevamente los argumentos y partes de la función

```
send.mail()
```

es la función que enviará el email

```
from="arigalli.unr.edu.ar"
```

cuenta desde la que se está enviando el mail

```
to=c("arigalli@unr.edu.ar","arigalli@conicet.gov.ar"),
```

cuenta a la que se envia el mail. Si tiene más de un destinatario, incluirlos en el paréntesis entre comillas separados por comas. Ejemplo

```
to=c("arigalli@unr.edu.ar","arigalli@conicet.gov.ar","sil_vaqero@hotmail.com","mala_lupo@hotm
ail.com")
```

Ya veremos formas más eficientes, especialmente cuando el listado es largo.

```
subject="mail desde ATLANTIS 3.1",
```

motivo del mail

```
body="cuerpo del mensaje",
```

mensaje que leerá el receptor

```
smtp=list(host.name="mail.unr.edu.ar",
```

dato obtenido de Details of selected server: Server Name

```
port=587,
```

dato obtenido de Details of selected server: Port

```
user.name="arigalli",
```

dato obtenido de Details of selected server: User Name

```
passwd="********",
```

dato que usted debe tener guardado y le permite comunicarse con el servidor. Esto cambiará para cada usuario.

```
ssl=FALSE,
```

dato obtenido de Details of selected server: Connection Security. En este caso es FALSE, ya que figura None

```
authenticate=TRUE,
```

permite autenticar la contraseña utilizada

+ send=TRUE)

envia el mail.

Envío de mails a lista de destinatarios

En el caso que el número de correos destinatarios sea elevado, es conveniente crear un objeto con los emails de los mismos. Básicamente, la creación de un vector al que por ejemplo podemos llamar "destinatarios" y colocar allí las direcciones.
Supogamos que tenemos que enviar los mails a dos direcciones como hemos visto: "arigalli@unr.edu.ar","arigalli@conicet.gov.ar"
creamos el vector

```
> destinatarios<-c("arigalli@unr.edu.ar","arigalli@conicet.gov.ar")
```

y en la función send.mail(), reemplazamos los mismos en el argumento to, por el vector creado, como se muestra resaltado en color amarillo.

```
> send.mail(from="arigalli@unr.edu.ar",to=destinatarios, subject="mail desde ATLANTIS 3.1", body="Este es un mail enviado por ATLANTIS para saludarlo por el día del biotecnólogo",smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="********", ssl=FALSE),authenticate=TRUE,send=TRUE)
```

Las direcciones de correo electrónico podrían hallarse en un data.frame. Por ejemplo disponemos de una base de datos de nombre y direcciones en un data.frame que llamamos "direcciones". A fines prácticos tenemos solo dos direcciones en la columna 2, llamada email, como muestra la Figura 3

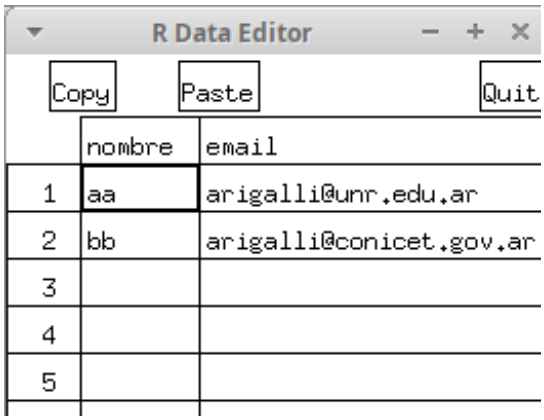


Figura 3

podemos reemplazar el argumento "to" por la columna del data.frame, quedando como se muestra a continuación

```
> send.mail(from="arigalli@unr.edu.ar",to=direcciones$email, subject="mail desde ATLANTIS 3.1", body="Este es un mail enviado por ATLANTIS para saludarlo por el día del biotecnólogo",smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="********", ssl=FALSE),authenticate=TRUE,send=TRUE)
```

Supongamos que la base de datos tiene una tercer columna, que clasifica a las personas como "profesional" y =tecnico", como muestra la .

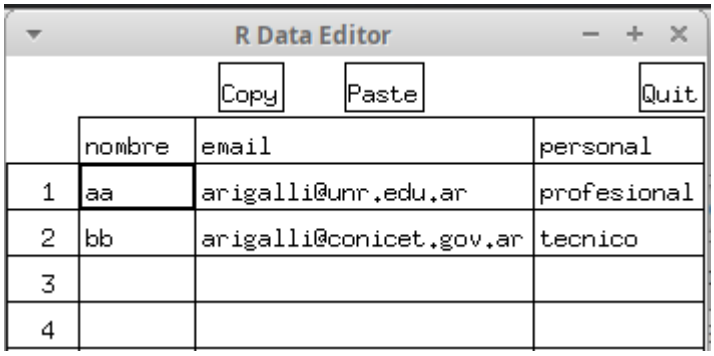


Figura 4

Puede enviar mails condicionando el envio según profesión. Supongamos que solo deseamos enviar el mail a los profesionales. El código quedaría como se indica a continuación

```
>
send.mail(from="arigalli@unr.edu.ar",to=direcciones$email[direcciones$personal=="profesional"], subject="mail desde ATLANTIS 3.1", body="Este es un mail enviado por ATLANTIS para saludarlo por el día del  biotecnólogo",smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="********", ssl=FALSE),authenticate=TRUE,send=TRUE)
```

Envío de mail con archivo adjunto

Si se desea enviar un archivo adjunto se utiliza la misma funcion send.mail(), con los mismos argumentos descriptos anteriormente, salvo que se agrega otro argumento: attach.files, en el que se indica la ruta y el nombre del archivo. En el caso que se muestra enviaremos un archivo adjunto de tipo txt que se halla en el Desktop de la computadora

```
>
send.mail(from="arigalli@unr.edu.ar",to=c("arigalli@unr.edu.ar","arigalli@conicet.gov.ar"),subject ="mail enviando archivo", body="Este mail lleva adjunto el archivo firma.txt",smtp=list(host.name="mail.unr.edu.ar",port=587,user.name="arigalli",passwd="******", ssl=FALSE),authenticate=TRUE,send=TRUE,attach.files="/home/alfredo/Desktop/firma.txt")
```

En la clase siguiente veremos el envío de mails desde la ejecución de scripts, estableciendo sistemas automáticos de información y alerta sobre dificultades.

Módulo 9 – clase 5

Envío automático de email con R

Ya hemos desarrollado como enviar un email desde un espacio de trabajo. El mail puede ser enviado a un destinatario o a una lista de destinatarios. También hemos desarrollado los argumentos de la función send.mail() del paquete mailR.
Hoy veremos un mecanismo automático que nos permita enviar el mail y otras funciones que sean de utilidad.
Plantearemos para esto un caso ficticio, pero que puede servir de ejemplo para que luego sea adaptado y utilizado.
Supongamos que usted es el director de un laboratorio que tiene 5 operarios.
1- Usted adjudica a cada operario la tarea a realizar cada día y el tiempo que dispone cada uno para realizarlo. Puede programar todos los días que desea con anticipación.
2- R envia a cada uno de ellos un mail informado la tarea que tiene que realizar y la cantidad de días disponibles para ejecutarla. Usted programa el sistema para que el mail sea enviado el día anterior a las tareas

3- Una vez que la ejecuta el operario ingresa a un script y carga la tarea realizada.
4- Pasado el período de tiempo de que dispone, el operario recibirá un aviso, en caso que no se haya completado la misma.

Construcción de la base de datos

El sistema funcionará con una base de datos que llamaremos **tareasoperarios**. Esta base será un data.frame que tendrá las siguiente columnas

fecha: por comodidad la pondremos en el formato producido por la función Sys.Date(), por ejemplo: 2023-03-16, por el 16 de marzo de 2023

nombre: apellido y nombre del operario. Por comodidad los llamaremos: AA, BB, etc

documento: numero de documento, pasaporte, etc, o cualquier otro número que no se repite de un operario a otro. Bien podría haber dos Juan Pérez, pero no habrá dos con el mismo documento. Por comodidad utilizaremos un solo dígito: 1, 2, 3, etc.

tarea: Protocolo a ejecutar. Suponemos que el laboratorio está organizado y todo lo que se hace está escrito paso a paso. Cada protocolo tiene un número, por ejemplo p1, p2, p3, etc. Los operarios saben donde hallar los mismos y qué hacer en cada caso. Su laboratorio cuenta con 187 protocolos estandarizados de trabajo.

tiempolimite: tiempo en días que tiene el operario para ejecutar la tarea, en función de las necesidades y

mail: llevará el mail del operario. Sugerimos en su ejemplo colocarles a todos su mail. De esa manera al ejecutar los pasos, todo le llegará a usted y podrá comprobar si las cosas funcionan correctamente.

tareacumplida: campo que completa el operario al terminar la misma. Cuando usted cargue la tarea puede adjudicarle el valor "no" y cuando el operario la ejecute, se reemplazará por "si"

Abrimos un espacio de trabajo. En este caso utilizaremos /media/alfredo/...../R95. Puede usted elegir el que quiera. Solo tendrá que tener la precaución de colocar los elementos que vayamos creando en ese sitio o donde corresponda.

Creamos entonces la base de datos, primero lo hacemos sin ningún registro

```
> tareasoperarios<-data.frame()
```

y la completaremos con la tarea para dos días consecutivos

```
> tareasoperarios<-edit(tareasoperarios)
```

La carga queda como muestra la Figura 1

R Data Editor

Copy Paste

	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	no
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	no
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	no
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	no
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no
6	2023-03-23	AA	1	p6	2	arigalli@unr.edu.ar	no
7	2023-03-23	BB	2	p7	3	arigalli@unr.edu.ar	no
8	2023-03-23	CC	3	p8	1	arigalli@unr.edu.ar	no
9	2023-03-23	DD	4	p9	1	arigalli@unr.edu.ar	no
10	2023-03-23	EE	5	p10	1	arigalli@unr.edu.ar	no
11	2023-03-24	AA	1	p11	1	arigalli@unr.edu.ar	no
12	2023-03-24	BB	2	p12	1	arigalli@unr.edu.ar	no
13	2023-03-24	CC	3	p13	1	arigalli@unr.edu.ar	no
14	2023-03-24	DD	4	p14	1	arigalli@unr.edu.ar	no
15	2023-03-24	EE	5	p15	10	arigalli@unr.edu.ar	no

Figura 1

Buen comienzo, ya sabemos qué día debe realizar cada uno la tarea asignada.
Seguramente tendrá que modificar fechas en el data.frame, para hacer las pruebas. También puede modificar la fecha de su sistema, pero lo desaconsejamos. Entonces en el data.frame, la primer fecha es 2023-3-22. Supongamos que hoy es 2023-3-21. Habría que avisar a los operarios qué tienen que hacer mañana.
Es importante para programa los mails automáticos recordar el número de cada columna. Lo vemos con

```
> names(tareasoperarios)
```

```
[1] "fecha"        "nombre"      "documento"    "tarea"
[5] "tiempolimite" "mail"        "tareacumplida"
```

Aviso automático de tarea a ejecutar

Podríamos pensar varios mecanismos para avisar a los operarios, pero se deben cuidar algunos detalles
1- Solo debe salir un mail indicando de la tarea a realizar en tal o cual fecha. Es decir que deberá filtrar las fechas, con algún criterio. Por ejemplo tomaremos que el mail salga el día anterior a la tarea.
2- Cada operario debe recibir su tarea y se le debe indicar el tiempo disponible
Generaremos un pequeño script que realice este procedimiento lo llamaremos **tareas.R**
Este script tiene que evaluar la fecha actual, compararla con la fecha de cada tarea y si falta un día enviar el mail. En la tabla siguiente vemos el código y su explicación. Dejaremos espacios para separar las rutinas y que no se superpongan con las explicaciones

código script: tareas.R	explicacion
library(mailR) library(rJava)	carga biblioteca mail.R carga biblioeca rJava

for(i in 1: nrow(tareasoperarios)){	recorre todo el data.frame tareasoperarios
if(as.numeric(strptime(as.character(Sys.Date()),"%Y-%m-%d")-strptime(tareasoperarios[i,1],"%Y-%m-%d"))== -1) {	Evalua si la diferencia entre la fecha actual y la correspondiente a la tarea es -1. Es decir que estamos en el día anterior a la tarea. Solo se ejecutará el envío de mail si se cumple esa condición
send.mail(from="arigalli@unr.edu.ar", to=tareasoperarios[i,6], subject=paste("tarea correspondiente al:" ,tareasoperarios[i,1], sep=""), body= paste("Buen día ", tareasoperarios[i,2], ". El día : ", tareasoperarios[i,1], ", le corresponde la tarea: ", tareasoperarios[i,4], ", para lo cual dispone de: ", tareasoperarios[i,5],sep=""),	envia el mail desde mi email utilizando el mail de cada fila indica la tarea del día El cuerpo del mail tendrá un mensaje por ejemplo: El 2023-03-31, le corresponde la tarea p2
smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="******", ssl=FALSE),authenticate=TRUE,send=TRUE) } }	parametros del servidor

Siga los pasos que se detallan
1- Ejecute
> Sys.Date()
2023-03-21
Tome el dato que le da.
2- Si es necesario modifique el data.frame: operariostareas de manera que la primer fecha para los 5 operarios sea un día posterior a su fecha y la segunda para los 5 operarios sea dos días posteriores. Como muestra la Figura 1. Estas coincidencias las hacemos para que pueda verificar el funcionamiento
3- Copie el código de la primer columna de la tabla anterior. Péguelo en un archivo de texto. Nombrelo como tareas.R y guardelo en el mismo espacio de trabajo que se halla el data.frame: tareasoperarios. Por supuesto podría trabajar con diferentes directorios e inclusive diferentes unidades de almacenamiento. Pero evitaremos complicaciones en este caso.
4- Ejecute
source('tareas.R')
5- abra su administrador de correos electrónicos. Si todo anduvo bien deberá haber recibido solo 5 mails personalizados para cada operario, indicando su tarea y fecha a realizar

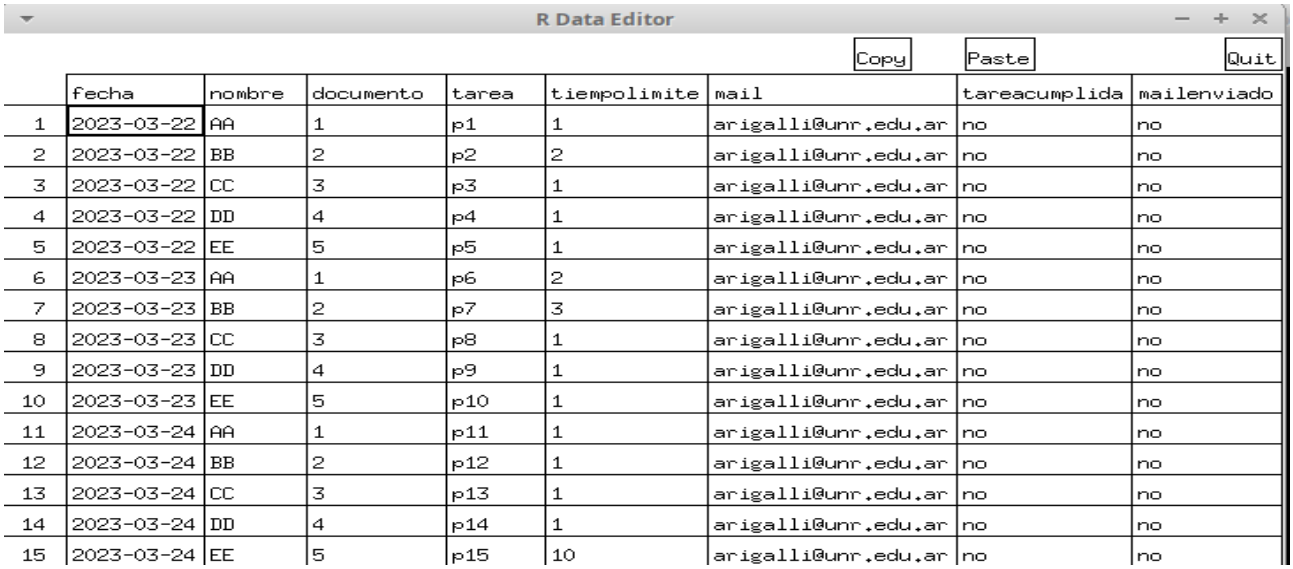
Optimizaciones

Optimización 1

Si bien el script muestra un muy buen funcionamiento, es evidente que si el mismo día se ejecuta nuevamente, volverá a mandar los mails. A nadie le gusta recibir mails repetidos y menos para recordarle sobre una tarea a realizar. Además, si se hace una costumbre, los mails comenzarán a ser ignorados por los interesados. Un solo mail es suficiente para avisar que hay que trabajar.
El mecanismo será sencillo. En la misma tabla tareasoperarios, se agregará una columna que tenga registrado si se envió el mail con dos opciones "no" y "si". Cuando se carga la tarea se colocará no, y el script colocará "si", al enviar el mail.
Corrijamos entonces. Editamos el data.frame

```
> tareasoperarios<-edit(tareasoperarios)
```

agregamos una columna enviomail. Colocamos **no** en la casilla correspondiente, ya que aun ninguna tarea fue informada y menos ejecutada, Figura 2



	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida	mailenviado
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	no	no
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	no	no
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	no	no
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	no	no
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no	no
6	2023-03-23	AA	1	p6	2	arigalli@unr.edu.ar	no	no
7	2023-03-23	BB	2	p7	3	arigalli@unr.edu.ar	no	no
8	2023-03-23	CC	3	p8	1	arigalli@unr.edu.ar	no	no
9	2023-03-23	DD	4	p9	1	arigalli@unr.edu.ar	no	no
10	2023-03-23	EE	5	p10	1	arigalli@unr.edu.ar	no	no
11	2023-03-24	AA	1	p11	1	arigalli@unr.edu.ar	no	no
12	2023-03-24	BB	2	p12	1	arigalli@unr.edu.ar	no	no
13	2023-03-24	CC	3	p13	1	arigalli@unr.edu.ar	no	no
14	2023-03-24	DD	4	p14	1	arigalli@unr.edu.ar	no	no
15	2023-03-24	EE	5	p15	10	arigalli@unr.edu.ar	no	no

Figura 2

Modificaremos el script para que cuando envie el mail en la columna 8, cambie **no** por **si**. De esta manera, al haber enviado el mail del 22/3, se modifica automáticamente el contenido de la columna, Figura 2

Haremos algunos cambios en el cuerpo del script, que llamaremos tareas1.R. En la condición if() agregaremos que se fije en la columna 8 y solo ingrese si además de la diferencia de tiempo, la columna 8 tenga el valor "no". Además debemos cambiar la condición de la columna enviomail. Se resaltan en amarillo los cambios realizados

código script: tareas1.R	explicacion
library(mailR) library(rJava)	carga biblioteca mail.R carga biblioeca rJava
for(i in 1: nrow(tareasoperarios)){	recorre todo el data.frame tareasoperarios
if((as.numeric(strptime(as.character(Sys.Date()),"%Y-%m-%d")-strptime(tareasoperarios[i,1],"%Y-%m-%d"))== -1) & tareasoperarios[i,8]=="no"){	Evalua si la diferencia entre la fecha actual y la correspondiente a la tarea es -1. Es decir que estamos en el día anterior a la tarea. Solo se ejecutará el envío de mail si se cumple esa condición. Además lo enviará si aun no se hizo. Si ya se envio, no lo volverá a hacer
send.mail(from="arigalli@unr.edu.ar", to=tareasoperarios[i,6], subject=paste("tarea correspondiente al:" ,tareasoperarios[i,1], sep=""), body= paste("Buen día ", tareasoperarios[i,2], ". El día : ", tareasoperarios[i,1], ", le corresponde la tarea: ", tareasoperarios[i,4], ", para lo cual dispone de: ", tareasoperarios[i,5],sep=""),	envia el mail desde mi email utilizando el mail de cada fila indica la tarea del día El cuerpo del mail tendrá un mensaje por ejemplo: El 2023-03-31, le corresponde la tarea p2
smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="******", ssl=FALSE),authenticate=TRUE,send=TRUE)	parametros del servidor
tareasoperarios[i,8]<-"si"	cambia el valor de la columna enviomail a "si", en la fila que se envió.
} }	Llave que cierra el if() Llave que cierra el for()

Ejecute los siguientes pasos

1- Copie el contenido de la primer columna, peguelo en un archivo de texto con el nombre tareas1.R y guardelo en el directorio que está su espacio de trabajo
2- Desde el espacio de trabajo ejecute
> source('tareas1.R')
Si la fecha de su computadora es 2023-03-21, cosa que puede comprobar con Sys.Date(). Debería recibir 5 mail en su casilla, cada uno para cada operario con detalles de la tarea.
3- Ejecute nuevamente el script. Podrá comprobar que a pesar que falta un día para la tarea no se envian más mails.
Podrá comprobar con
> edit(tareasoperarios)
que ahora la columna 8 para las tareas del 2023-03-17 figura con si, ya que se envio el mail, Figura 3

R Data Editor

Copy Paste Quit

	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida	mailenviado
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	no	si
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	no	si
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	no	si
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	no	si
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no	si
6	2023-03-23	AA	1	p6	2	arigalli@unr.edu.ar	no	no
7	2023-03-23	BB	2	p7	3	arigalli@unr.edu.ar	no	no
8	2023-03-23	CC	3	p8	1	arigalli@unr.edu.ar	no	no
9	2023-03-23	DD	4	p9	1	arigalli@unr.edu.ar	no	no
10	2023-03-23	EE	5	p10	1	arigalli@unr.edu.ar	no	no
11	2023-03-24	AA	1	p11	1	arigalli@unr.edu.ar	no	no
12	2023-03-24	BB	2	p12	1	arigalli@unr.edu.ar	no	no
13	2023-03-24	CC	3	p13	1	arigalli@unr.edu.ar	no	no
14	2023-03-24	DD	4	p14	1	arigalli@unr.edu.ar	no	no
15	2023-03-24	EE	5	p15	10	arigalli@unr.edu.ar	no	no

Figura 3

Optimización 2

Según el sistema operativo que utilice y los recursos disponible, podrá organizar para que el script se ejecute automáticamente cada día. Para esto dejamos librado a su imaginación el procedimiento. Pero proponemos algunas ideas
1- Si la pc queda encendida las 24 hs, el script podría permanecer en ejecución con un contador de horas o registro de fecha, para que cada día cuando se cumpla una dada condición, se envie el mail
2- si la pc no queda encendida. Se pude hacer un acceso a través de las tareas que se ejecutan al iniciar la computadora, colocando allí el script.
Seguramente su imaginación podrá agregar otros mecanismos posibles.

Cumplimiento de tareas por los operarios

Ya estamos en el día 2023-03-22, que se deben cumplir tareas. El operario accederá a través de un script al espacio de trabajo y reportará su tarea.

El script tiene algunas simplificaciones con el fin de hacer el ejemplo más didáctico. Todo es perfeccionable en el ambiente de scripts.
Analicemos el script que llamamos cumplimiento.R

código script: cumplimiento.R	explicacion
print('Script para reportar ejecución de tarea') print('La lista indica documentos, tareas pendientes y dias disponibles para su ejecución')	informa sobre la acción da una lista de documentos, tareaspendientes y dias para ejecución
print(tareasoperarios[tareasoperarios$tareacumplida=="no" & as.numeric(strptime(as.character(Sys.Date()),"%Y-%m-%d")-strptime(tareasoperarios$fecha,"%Y-%m-%d"))== 0,c(3,4,5)])	Imprime en pantalla los documentos, tareas y dias, pero solo los que corresponden al día de la fecha
print('Ingrese su número de documento') documento<-as.numeric(scan(file="",what="",nmax=1)) print('Ingrese su tarea cumplida') tarea<-scan(file="",what="",nmax=1)	pide el documento del operario pide la tarea que cumplió

for(i in 1:nrow(tareasoperarios)){
if((as.numeric(strptime(as.character(Sys.Date()),"%Y-%m-%d")-strptime(tareasoperarios[i,1],"%Y-%m-%d"))>= 0) & tareasoperarios[i,7]=="no" & tareasoperarios[i,4]==tarea & tareasoperarios[i,3]==documento)
tareasoperarios[i,7]<-"si"
}
	recorre todo el data.frame
	Estructura if que colocara "si" en la columna tarea cumplida si la diferencia de fecha entre hoy la fecha de tarea es mayor o igual a cero, si la tarea aun no fue cumplida, si la tarea coincide con la introducida por el operario y el documento coincide con el valor introducido

readline("su tarea ha sido registrada, oprima enter para continuar")	Avisa que fue registrada la ejecución de la tarea

Probemos el script, pero antes veamos la tabla con las tareas al día de hoy

Copy Paste Quit

	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida	mailenviado
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	no	si
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	no	si
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	no	si
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	no	si
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no	si
6	2023-03-23	AA	1	p6	2	arigalli@unr.edu.ar	no	no
7	2023-03-23	BB	2	p7	3	arigalli@unr.edu.ar	no	no
8	2023-03-23	CC	3	p8	1	arigalli@unr.edu.ar	no	no
9	2023-03-23	DD	4	p9	1	arigalli@unr.edu.ar	no	no
10	2023-03-23	EE	5	p10	1	arigalli@unr.edu.ar	no	no
11	2023-03-24	AA	1	p11	1	arigalli@unr.edu.ar	no	no
12	2023-03-24	BB	2	p12	1	arigalli@unr.edu.ar	no	no
13	2023-03-24	CC	3	p13	1	arigalli@unr.edu.ar	no	no
14	2023-03-24	DD	4	p14	1	arigalli@unr.edu.ar	no	no
15	2023-03-24	EE	5	p15	10	arigalli@unr.edu.ar	no	no

Figura 4
Como muestra la Figura 4, hoy 2023-03-22, hay 5 tareas asignadas y ninguna está cumplida.
Supongamos que el operario AA ha cumplido su tarea, entonces procederá a registrarla.

Se dirigirá a una computadora preestablecida y ejecutará el script: cumplimiento.R, mostrando una pantalla como lo indica la Figura 5

```
> source('cumplimiento.R')
[1] "Script para reportar ejecución de tarea"
[1] "La lista indica documentos, tareas pendientes y dias disponibles para su ejecución"
  documento tarea tiempolimite
1         1    p1            1
2         2    p2            2
3         3    p3            1
4         4    p4            1
5         5    p5            1
[1] "Ingrese su número de documento"
1:
```

Figura 5

sigue las instrucciones de la pantalla y al completar sus datos finalizará como muestra la Figura 6

```
> source('cumplimiento.R')
[1] "Script para reportar ejecución de tarea"
[1] "La lista indica documentos, tareas pendientes y dias disponibles para su ejecución"
  documento tarea tiempolimite
1         1    p1            1
2         2    p2            2
3         3    p3            1
4         4    p4            1
5         5    p5            1
[1] "Ingrese su número de documento"
1: 1
Read 1 item
[1] "Ingrese su tarea cumplida"
1: p1
Read 1 item
su tarea ha sido registrada, oprima enter para continuar
```

Figura 6

Si observamos el data.frame: tareasoperarios
> edit(tareasoperarios)

	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida	mailenviado
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	si	si
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	no	si
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	no	si
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	no	si
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no	si
6	2023-03-23	AA	1	p6	2	arigalli@unr.edu.ar	no	no
7	2023-03-23	BB	2	p7	3	arigalli@unr.edu.ar	no	no
8	2023-03-23	CC	3	p8	1	arigalli@unr.edu.ar	no	no
9	2023-03-23	DD	4	p9	1	arigalli@unr.edu.ar	no	no
10	2023-03-23	EE	5	p10	1	arigalli@unr.edu.ar	no	no
11	2023-03-24	AA	1	p11	1	arigalli@unr.edu.ar	no	no
12	2023-03-24	BB	2	p12	1	arigalli@unr.edu.ar	no	no
13	2023-03-24	CC	3	p13	1	arigalli@unr.edu.ar	no	no
14	2023-03-24	DD	4	p14	1	arigalli@unr.edu.ar	no	no
15	2023-03-24	EE	5	p15	10	arigalli@unr.edu.ar	no	no

Figura 7

comprobamos que el operario AA, documento: 1 ya tiene su tarea registrada.

Si AA volviera a ingresar al script, no le mostraría más su tarea, ya que no está pendiente, como muestra la Figura 8.

```
> source('cumplimiento.R')
[1] "Script para reportar ejecución de tarea"
[1] "La lista indica documentos, tareas pendientes y dias disponibles para su ejecución"
  documento tarea tiempolimite
2         2    p2            2
3         3    p3            1
4         4    p4            1
5         5    p5            1
[1] "Ingrese su número de documento"
```

Figura 8

Por supuesto que se pueden hacer cientos de modificaciones en el script para optimizar su función, peor no es el objetivo de la clase.

Mañana 2023-03-23, habrá tareas que tendrán el plazo vencido. El script debería avisar por mail que ha ocurrido el vencimiento.

Supongamos que los operarios de documento 2, 3 y 4 realizaron su tarea de manera correcta, pero el operario 5 no cumplió o no registró la tarea p5. Al final del día el data.frame tareasoperarios quedará como muestra la .

Copy Paste Quit

	fecha	nombre	documento	tarea	tiempolimite	mail	tareacumplida	mailenviado
1	2023-03-22	AA	1	p1	1	arigalli@unr.edu.ar	si	si
2	2023-03-22	BB	2	p2	2	arigalli@unr.edu.ar	si	si
3	2023-03-22	CC	3	p3	1	arigalli@unr.edu.ar	si	si
4	2023-03-22	DD	4	p4	1	arigalli@unr.edu.ar	si	si
5	2023-03-22	EE	5	p5	1	arigalli@unr.edu.ar	no	si

Figura 9

Entocnes el día 2023-03-23, se deberá avisar al operario EE, su atraso en la ejecución. Para simplicidad haremos un script aparte que llamaremos atraso.R.

Por supuesto que a los fines de optimización y orden será conveniente realizar un solo script para todo lo que venimos ejecutando o bien como se mostró en el módulo 5. Generar un script general que llame a cada uno de los scripts correspondientes. Como siempre en estos temas hay casi infinitas formas de hacerlo y cada uno irá eligiendo la que más le convenga y comprenda. Lo importante en los scripts no es tanto la elegancia, sino la eficiencia y la confianza que le tengamos.

Aviso de vencimiento de tarea

El script atraso.R, debe chequear cada día, los días anteriores. Si la tarea no está complida verificar si el tiempo es aun disponible. Por ejemplo hoy 2023-03-23, la tarea del operario EE estaría vencida, ya que disponia de 1 día, sin embargo la tarea del operario BB, no lo estaría ya que fue cargada el 22/3, pero dispone de dos días. Analicemos el script

atraso.R	explicacion
library(mailR) library(rJava)	carga biblioteca carga biblioteca
for(i in 1: nrow(tareasoperarios)){	Recorre la base de datos desde la primera a la última línea.
if(tareasoperarios[i,7]=="no"){	En cada linea se fija si la tarea no fue cumplida.
if(as.numeric(strptime(as.character(Sys.Date()),"%Y-%m-%d")-strptime(tareasoperarios[i,1],"%Y-%m-%d"))>= tareasoperarios[i,5]){	Si no fue cumplida se fija si la diferencia entre la fecha actual y la de la tarea es mayor o igual que el tiempo disponible
send.mail(from="arigalli@unr.edu.ar",	Si se cumple la condición anterior envia el mail desde nuestra dirección
to=tareasoperarios[i,6],	A la dirección de la línea analizada
subject=paste("tarea ATRASADA del:" ,tareasoperarios[i,1], sep=""),	Coloca un subject indicando el atraso
body= paste("Buen día ", tareasoperarios[i,2], ". Su tarea correspondiente al día : ", tareasoperarios[i,1], ", no ha sido realizada en fecha",sep=""),	Coloca un cuerpo del mail con detalles del incumplimiento
smtp=list(host.name="mail.unr.edu.ar", port=587, user.name ="arigalli",passwd="ac2371fg", ssl=FALSE),authenticate=TRUE,send=TRUE)	Parámetros del servidor
}	llave que cierra segundo if()
}	llave que cierra primer if()
}	llave que cierra for()

Este script enviará cada día un mail a los operarios atrasados, no importa cuantos dias lleven atrasados, ya que recorre toda la base de datos y evalua si está atrasado.
Como dijimos, el sistema creado requiere múltiples optimizaciones. Una de ellas es que si el operario recibe el mail indicando el atraso, quizás realice la tarea. Cuando ingrese a cargarla, no lo podrá hacer, ya que el script cumplimiento.R, solo muestra el día de la fecha.
Como siempre estos sistemas tienen optimización infinita y la misma es conveniente hacerla con el uso cotidiano, escuchando a los usuarios buscando claridad en la ejecución y la forma de mostrar sus resultados.

Módulo 9 – clase 6

Comunicación de R por puerto serial

Las siguientes clases de este módulo se enfocarán en la comunicación de un instrumento de medición con R. El desarrollo de las clases requiere conocimientos básicos de sistemas operativos, electrónica y programación. Puede ocurrir, que usted no tenga esos conocimientos, sin embargo las clases servirán para mostrarle un camino que no siendo sencillo es posible de transitar. El manejo de este tipo de recursos permite la construcción de instrumentos de medición con inversión mínima y funcionamiento adaptable a su necesidad. La combinación de estos conocimientos con el uso de microcontroladores programables, como es el caso de Arduino, hará de su vida una experiencia diferente.

Ya sabemos entonces: de acá en adelante no será fácil! Ya que no será fácil, comencemos los antes posible introduciendo ejemplos sencillos y que tendrán grandes simplificaciones. En algunos casos, estas simplificaciones podrán resultar ridículas a un ingeniero electrónico, otras veces a un bioquímico, también podrán sorprender a un especialista en sistemas y programación. La idea es hacer una explicación que pueda ser comprendida por todos, aceptando las simplificaciones en nuestras áreas de experticia.

Comunicación de un dispositivo con R

Para poder comunicar un dispositivo, por ejemplo un espectrofotómetro, un termómetro o sensor de movimiento, con R en una computadora debemos tener ciertos elementos

1- Un instrumento de medición con salida de señal que pueda llevarse a una computadora.

2- digitalizador de la señal, en caso que no lo sea.

3- comunicación de dispositivo digitalizador con R.

Instrumento de medici**ón y s**alida de señal

Para poder comprender esto, veamos una simplificación de un espectrofotómetro. Nuestros ojos y sistema nervioso es un buen espectrofotómetro. Por ejemplo si estamos frente a tres tazas con café, si las tazas son de vidrio transparente mejor, aquella taza que tenga color negro más tenue y transparente, sabemos que tiene menos café. Contrariamente, si queremos tomar un café bien fuerte, elegiremos la taza con el color negro más intenso. La espectrofotometría hace eso, asociar la concentración de una sustancia con la intensidad de su color.

En esta clase utilizaremos un viejo espectrofotómetro del CUEM, pero que gracias a la tecnología que desarrollamos, ha regresado para quedarse y brindar servicios. Este espectrofotómetro tiene como elemento de lectura una sistema analógico representado por una aguja móvil que mide valores de transmitancia de 0 a 100, Figura 1.

Figura 1

En general en las mediciones espectrofotométricas, la transmitancia es 100% cuando la solución no tiene color y es 0% o cercana a este valor cuando la solución es intensamente coloreada y no deja pasar nada de luz.
Breve explicación de funcionamiento de un espectrofotómetro. Estos instrumentos permiten medir la concentración de una sustancia en solución, en este caso la llamamos S. Para ello se preparan soluciones estándar que consisten en soluciones de concentración conocida de la sustancia S, las cuales tienen diferente color. Supongamos que a más concentración, tienen mayor color rojo, como muestra la Figura 2. Podemos ver que a concentracion 0 mg/L es transparente y a concentración 6 mg/L es rojo intenso. Si la solución tiene concentración entre 0 y 6 tendrá diferente tonalidades intermedias. A simple vista podríamos inferir cual tiene más o menos sustancia .... similar a lo que hicimos con la taza de café!!
Pero por qué usar entonces un instrumento como un espectrofotómetro? La respuesta es sencilla, nuestro ojo y sistema nervioso solo pueden comparar y decir tiene más o menos que tanto... pero el espectrofotómetro le puede poner un número al que llamamos transmitancia y abreviamos T%.

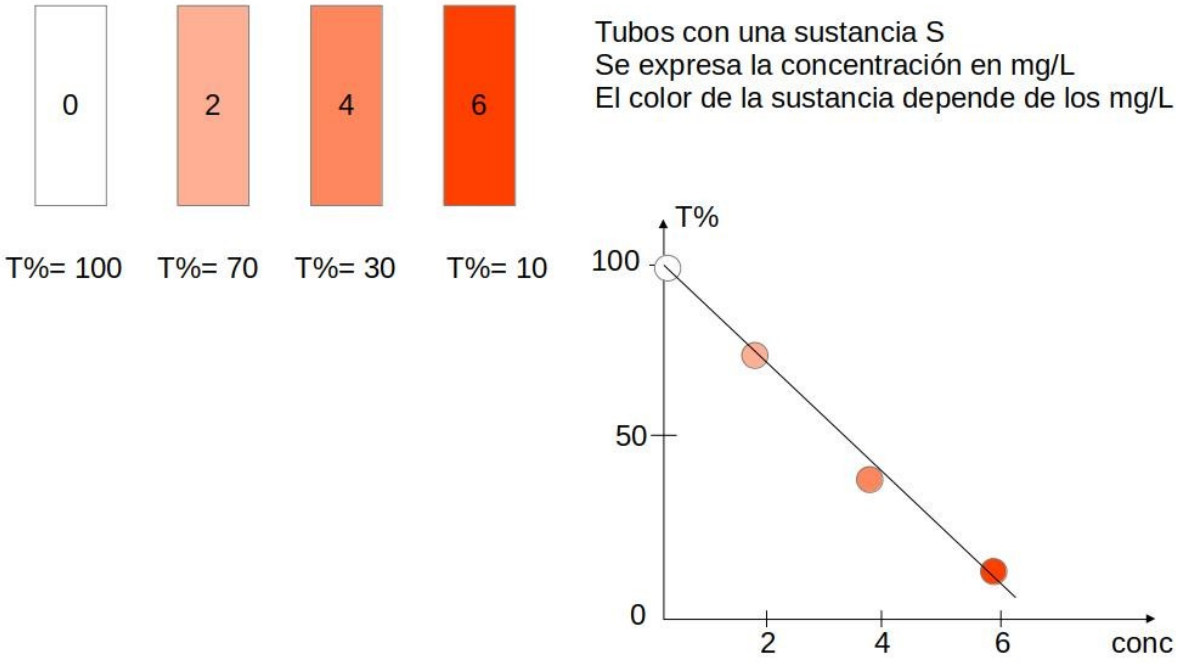


Figura 2

Asi, supongamos que en nuestro ejemplo las transmitancias para 0, 2, 4 y 6 mg/L fueron 100, 70, 30 y 10%. Como tenemos dos variables que se mueven con cierta correlación, en este caso a mayor concentración menor %T, podemos hacer una gráfica bidimensional y ajustar un modelo, como vimos en los módulos 3-7 de este curso, Figura 2.

Luego si tenemos un tubo con la sustancia S, pero de concentración desconocida, podemos medirle el %T en el mismo espectrofotómetro y la aguja nos mostrará un valor de %T, con el que realizaremos la interpolación con nuestro modelo para saber su concentración. En el ejemplo de la Figura 3, vemos un tubo cuya concentración es desconocida (??), medimos su %T, interpolamos con la curva de nuestro modelo (que se conoce habitualmente como curva de calibración) y sacamos su concentración. En este caso supongamos el valor es 5 mg/L

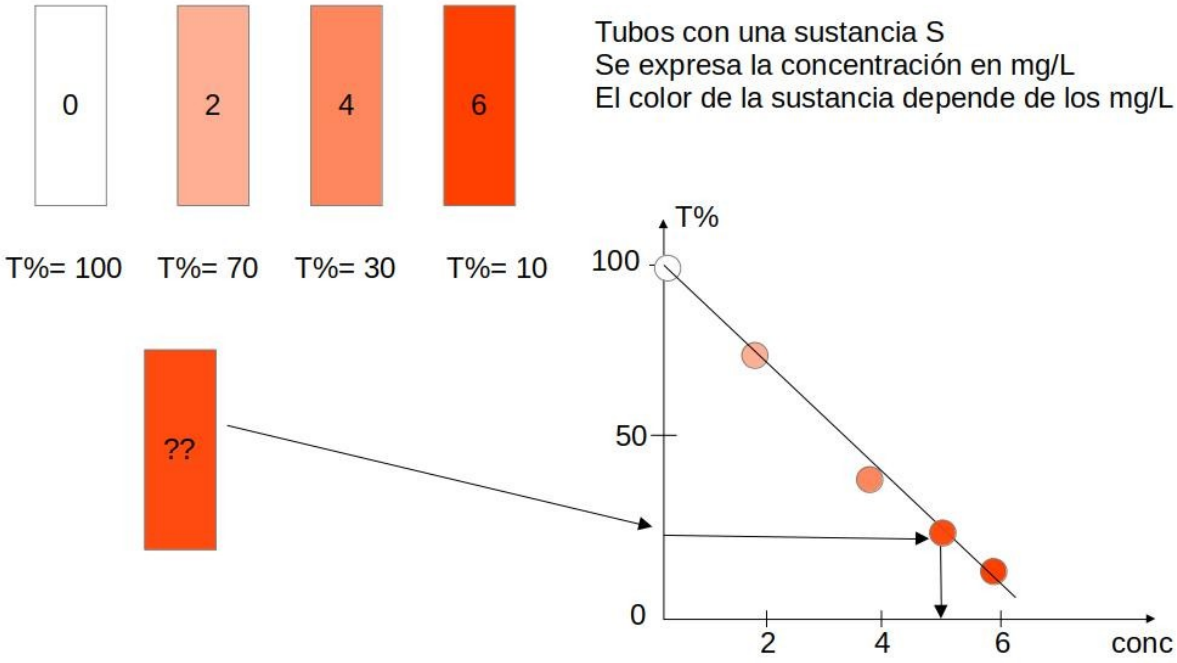


Figura 3

Para que este espectrofotómetro nos sirva para poder ingresar datos a una computadora y finalmene procesarlo con R, debería tener una salida que genere una señal eléctrica proporcional al %T. Afortunadamente el viejo Turner la tiene y da una señal de 0V cuando la %T=0 y 1V cuando %T=100. V representa a la unidad de la magnitud voltaje. En general en estos casos la señal que se toma es una señal eléctrica que corresponde a una diferencia de potencial que se mide en Volts.
Conclusión: si nuestro instrumento tiene una salida que indique diferentes voltajes para diferentes %T es un buen comienzo.
Esta señal es una señal que llamamos analógica, es decir, a mayor color menor %T y menor voltaje, y la variación de las dos últimas magnitudes es continua. Debemos digitalizar esa señal, es decir transformarle en valores discretos, que así lo entenderá nuestra computadora.
Digitalizar nuestros datos sería transformar las mediciones, al igual que nuestra curva en valores discretos, lo que generaría una curva de calibración como muestra la Figura 4. Es decir %T y concentración varían a saltos.

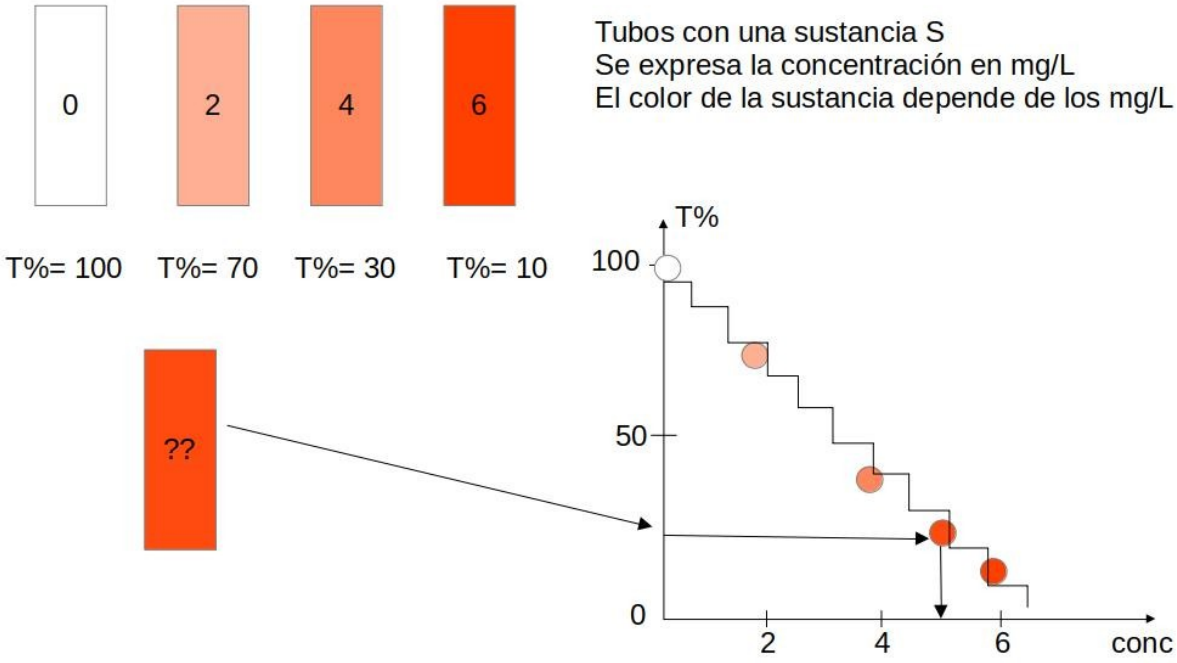


Figura 4

La cantidad y la magnitud de saltos dependerá del número de bits de nuestro sistema de comunicación.

Una analogía para comprender al menos el concepto del "número de bits". Lo haremos con las tazas de café y nuestro ojo-sistema nervioso.

Supongamos que nuestro ojo solo puede distinguir si una taza: tiene café (le asignamos 1) o no tiene café (le asignamos el valor 0). Diriamos que nuestro sistema de detección de café es de 1 bit: solo puede tener dos valores.

En cambio si otra persona es capaz de distinguir 4 grados de concentración de café con solo mirarlo diremos que su ojo es de 2 bits.

Si un tomador de café muy experimentado pudiera reconocer 8 concentraciones diferentes, diremos que su ojo es de 3 bits.

Si analizamos los números tenemos

2 concentraciones = 1 bit. Entonces podemos decir que el númeo de concentraciones es 2^1

4 concentraciones = 2 bits. Que equivale a 2^2

8 concentraciones = 3 bits: que equivale a $3^{3.}$

Como podemos ver el número de escalones que podemos discriminar es la base 2, elevada al número de bit. Si tuvieramos un ojo de 8 bits, quería decir que puedo distinguir entre 2^8 concentraciones diferentes, es decir: 2*2*2*2*2*2*2*2= 256.

Queda claro que cuantos más bits tenga un sistema de comunicación, mejor discriminará los valores medidos.

Digitalización de la información

Los equipos de medición pueden generar señales analógicas o digitales. Es claro que si la señal generada es digital, ya tendremos un problema resuelto. En caso que la señal sea analógica, existen diferentes dispositivos electrónicos que permiten digitalizar una señal. Nosotros trabajaremos con Arduino. Este dispositivo tiene puertos por los que ingreso una señal analógica y la transforma en digital, permitiendo luego el flujo de esa información

Comunicación dispositivo con R

Arduino se puede comunicar con una computadora a través de lo que se conoce como puerto serial. El puerto serial es una vía de comunicación entre un dispositivo (teclado, mouse, instrumento de medición) y el sistema operativo de una computadora. El puerto serial es una vía de comunicación bit a bit entre los dispositivos mencionados.
Este curso está basado en el sistema operativo Linux, por lo cual se darán las mayores explicaciones. Sin embargo se intentará brindar alguna ayuda respecto de su uso en Windows.
Puerto seriales en Linux

Los puertos seriales en Linux se identifican con las letras ttySX, donde X es un número. Por ejemplo ttyS0, ttyS1, etc.
Para la identificación de los puertos seriales, desde una consola ejecutamos el comando
dmesg | grep tty

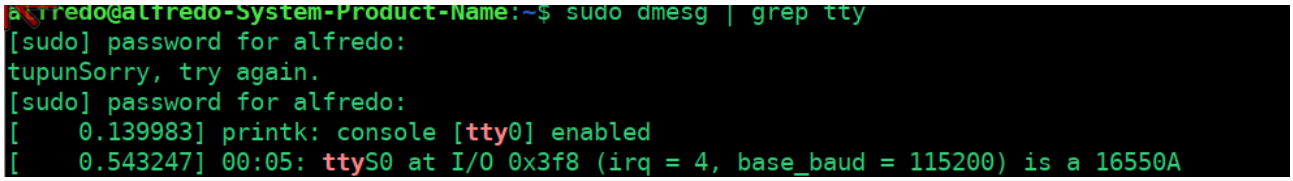


Figura 5
Vemos que tenemos el puerto ttyS0 con una velocidad de comunicación de 115200 baudios.
Otra forma para hacerlo es utilizando el comando
sudo setserial -g /dev/ttyS*

Si no tuviera instalada setserial, puede hacerlo directamente desde la consola con el comando
sudo apt-get install setserial
Una vez instalada ejecute
sudo setserial -g /dev/ttyS*

y podrá ver los puertos seriales intalados.

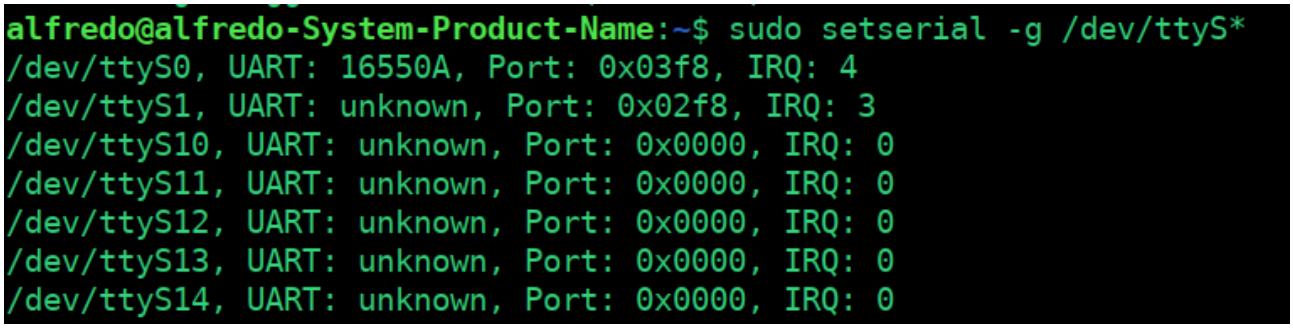


Figura 6

Por falta de experiencia no podemos mostrarle como acceder a puertos seriales en sistema operativo Windows. Esperamos en próxima edición de este módulo brindar dicha información.
Por este puerto ttyS0, nos comunicaremos con un instrumento, pero para hacerlo con R deberemos instalar el paquete serial de R.
La biblioteca serial

Descargue esta biblioteca de algún repositorio como lo hemos hecho en todo el curso
> install.packages("serial",dependencies=TRUE)

y luego cárguelo en su espacio de trabajo
> library(serial)

Veremos en esta clase alguno comandos de esta biblioteca
Comando listPorts()

```
> listPorts()
```

Hint: On unix-systems the port list might be incomplete!
[1] "ttyS0"

Como vemos nos da la misma información que los dos comandos utilizados desde consola de Linux.
Una vez conocido el puerto serial y su velocidad creamos el objeto con, para luego abrir la conexión. Esto lo logramos con el comando serialConnection, donde utilizaremos el puerto, en este caso ttyS0 y su velocidad en el argumento mode: 115200. Los otros parámetros los dejamos como figura en la línea sigueinte

```
> con <- serialConnection(name = "testcon",port = "ttyS0",mode = "115200,n,8,1",newline = 1,translation = "crlf")
```

luego abrimos la con el siguiente comando

```
> open(con)
```

Error: [tcl] couldn't open "/dev/ttyS0": permission denied.> open(con)
si obtuvimos el mensaje de Error mostrado es porque debemos habilitar la posibilidad de acceso, lectura y escritura del archivo ttyS0
Para ello desde una consola ejecutamos
sudo chmod 666 /dev/ttyS0
nos pedirá contraseña y oprimimos enter. Ahora podremos ejecutar el comando open()

```
> open(con)
```

vemos que no nos da el mensaje de error.
probamos si la conexión está abierta con el comando isOpen()

```
> isOpen(con)
```

[1] TRUE
como podemos ver con open(con), la hemos dejado abierta
si deseamos cerrar la conexión serial lo podemos hacer con el comando close()

```
> close(con)
```

y si ejecutamos nuevamente isOpen()

```
> isOpen(con)
```

[1] FALSE
comprobamos que la hemos cerrado.

Para seguir adelante le recomendamos adquirir algunos componentes. Las figuras siguientes son específicamente lo que se requerirá para hacer el proyecto y los precios mostrados corresponden a valores actuales (abril 2023) del mercado.
Utilizaremos una placa de Arduino UNO, que puede hacer a través de internet. Verá que hay versiones más económicas que corresponden a clones de Arduino, ya que Arduino es una plataforma de hardware y software libre. No tenemos suficiente experiencia con muchas versiones, por lo que recomendamos el Arduino UNO indicado.

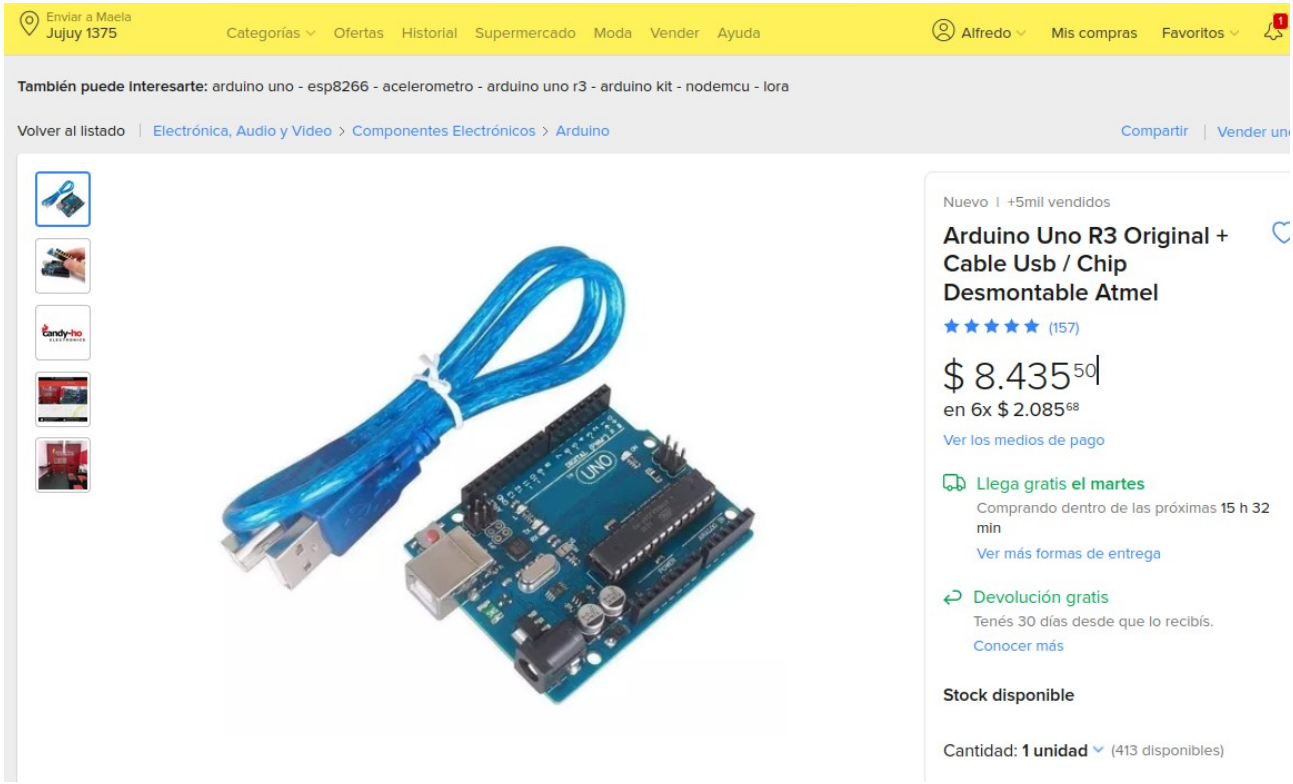


Figura 7

un sensor de temperatura DHT11

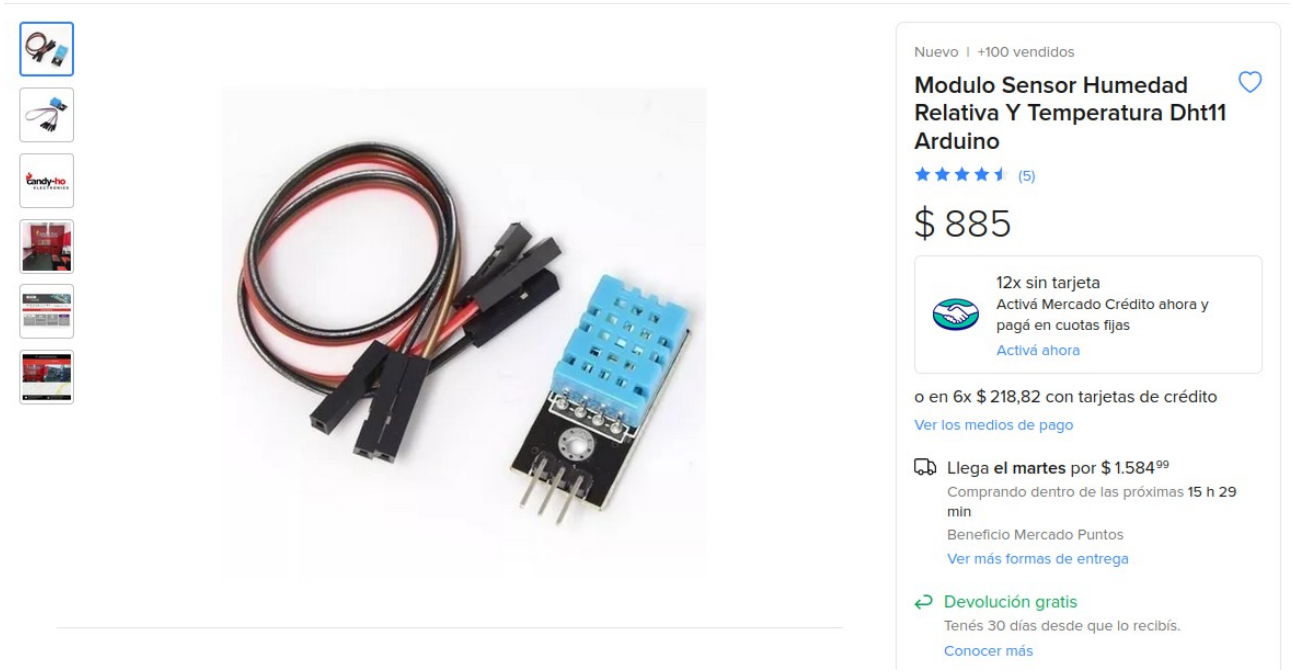


Figura 8

Para simplificar el trabajo posterior recomendamos la adquisición de los siguientes cables de conexión

Figura 9

Con esos elementos podremos hacer la primer prueba de conexión entre un sensor, Arduino y R.

Módulo 9 – clase 7

Arduino

Para quienes son usuarios de R, Arduino no será tan dificultoso, luego de dar los primeros pasos. Hay una similitud entre ellos que con el uso de ambos se irá notando cada día más.

Hagamos una breve descripción de R y de Arduino, aceptando grandes simplificaciones.

R es un entorno útil para el almacenamiento y análisis de datos que se fundamenta en el uso de funciones que realizan tareas específicas. Estas funciones se hallan dentro de bibliotecas que en algunos casos se cargan automáticamente y en otros casos deben ser cargadas en el espacio de trabajo cuando se las desea.

Arduino es una plataforma de desarrollo electrónico que permite recibir información del exterior. Esta información es recibida por funciones que pertenecen a bibliotecas especiales que deben ser incorporadas habitualmente al espacio de trabajo.

Ambos entornos necesitan de un lenguaje de programación. En R lo utilizamos cuando comenzamos a desarrollar scripts, situación en que seguro se hallan todos los alumnos de este curso. Arduino tiene una plataforma de desarrollo en que se escribe un programa en lenguaje C, que como verá, es similar al utilizado en R.

Cuando haya finalizado su primer proyecto en Arduino, cosa que hará en este módulo, verá un nuevo horizonte en su trabajo de investigación!

No desespere ni se desanime en la primer lectura e intento de montaje de los componente, instalación de bibliotecas y escritura del sketch.

En esta clase realizará trabajos que implican electricidad. Nada superará los 5 V por lo cual puede manejarlo sin ningún riesgo. También realizará conexiones entre componentes

electrónicos, en este caso relativamente económicos. Si conectará algo erroneamente, normalmente la consecuencia será que no funcionará o lo hará de manera no óptima. Rara vez ocurre daño de los componentes. Tampoco debemos abusar o conectar erroneamente, para ver los resultados. El riesgo de daño de los componentes es posible, pero raro.
Si nunca ha realizado trabajos en electrónica, adelante. Un nuevo horizonte se abre para usted!!!
Qué necesitamos para trabajar con Arduino y R?

Entorno de R

Si ha llegado hasta acá, no necesita esta explicación. R se instala en cualquier computadora con mínimos requerimientos, como se ha mostrado en el módulo 1. Con R instalado y la capacidad de descargar bibliotecas de algún repositorio es suficiente.
Entorno Arduino

Instalar la interfaz de trabajo Arduino
1- dirigirse a la siguiente dirección
https://www.arduino.cc
Hallará la imagen de la Figura 1

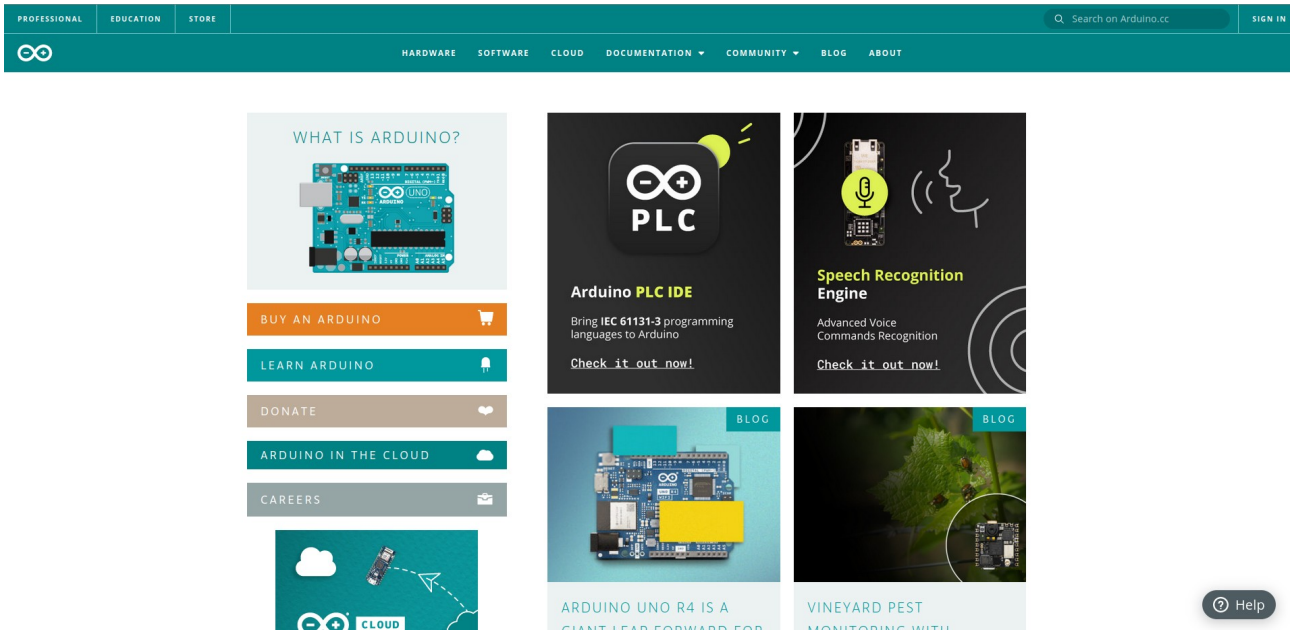


Figura 1

2- haga clic en SOFTWARE, con lo que llegará a la página de la Figura 2

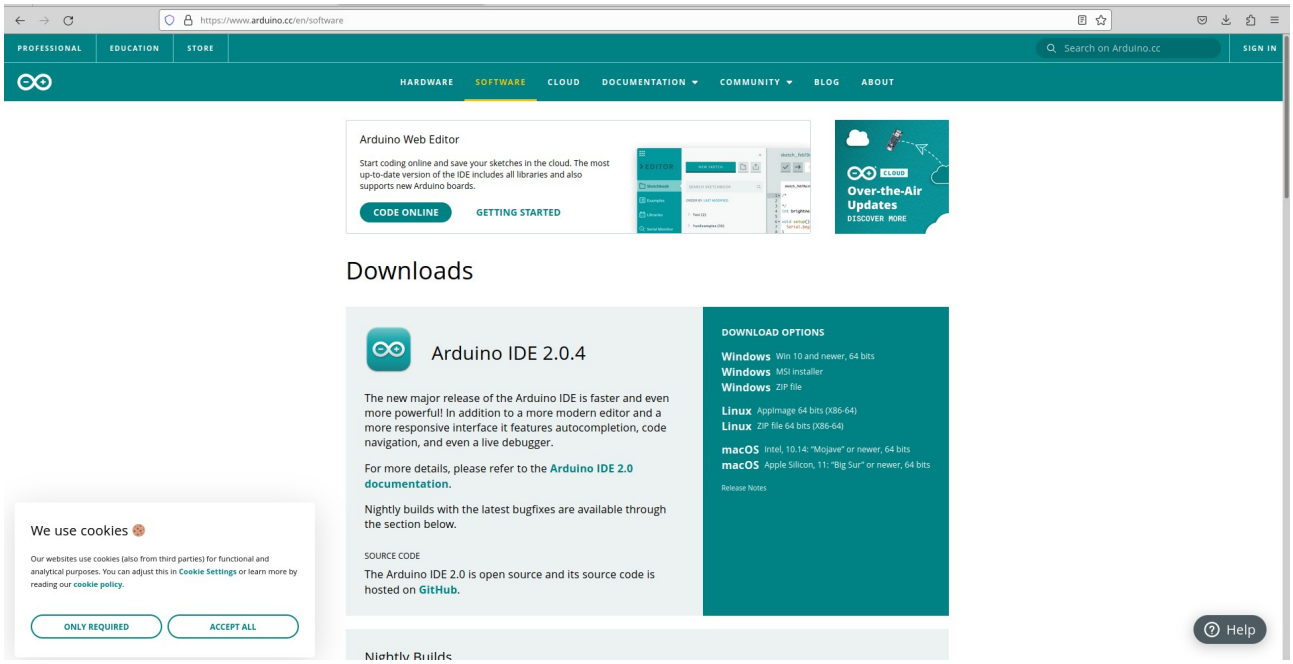


Figura 2

3- Descargue la versión que corresponde según el sistema operativo, eligiendo del recuadro DOWNLOAD OPTIONS. Así llegará a una página en la que podemos hacer una donación o bien descargarlo libremente oprimiendo JUST DOWNLOAD, Figura 3

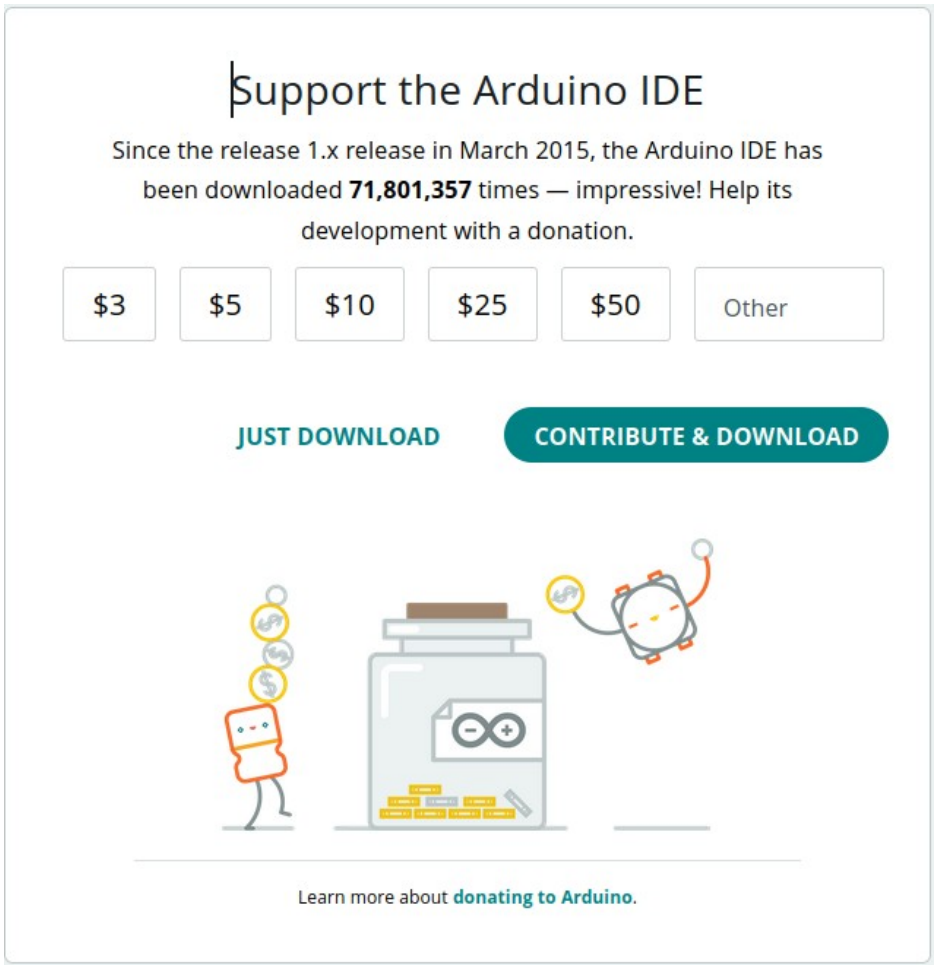


Figura 3

Instalación en Linux

Si eligió la opción para Linux y optó por el archivo de extensión .zip, se descargará la versión de arduino en la carpeta Download en formato .zip. Haciendo doble clic sobre el archivo arduino-ide_2.0.4_Linux_64bit.zip, se extraerán los archivos en la carpeta que usted elija, en este caso utilizamos la carpeta Arduino dentro de una de las unidades de almacenamiento, Figura 4. Allí se ha creado una carpeta: arduino-ide_2.0.4_Linux_64bit, en la que vemos varios archivos, entre ellos arduino-ide que es el archivo ejecutable que utilizaremos.

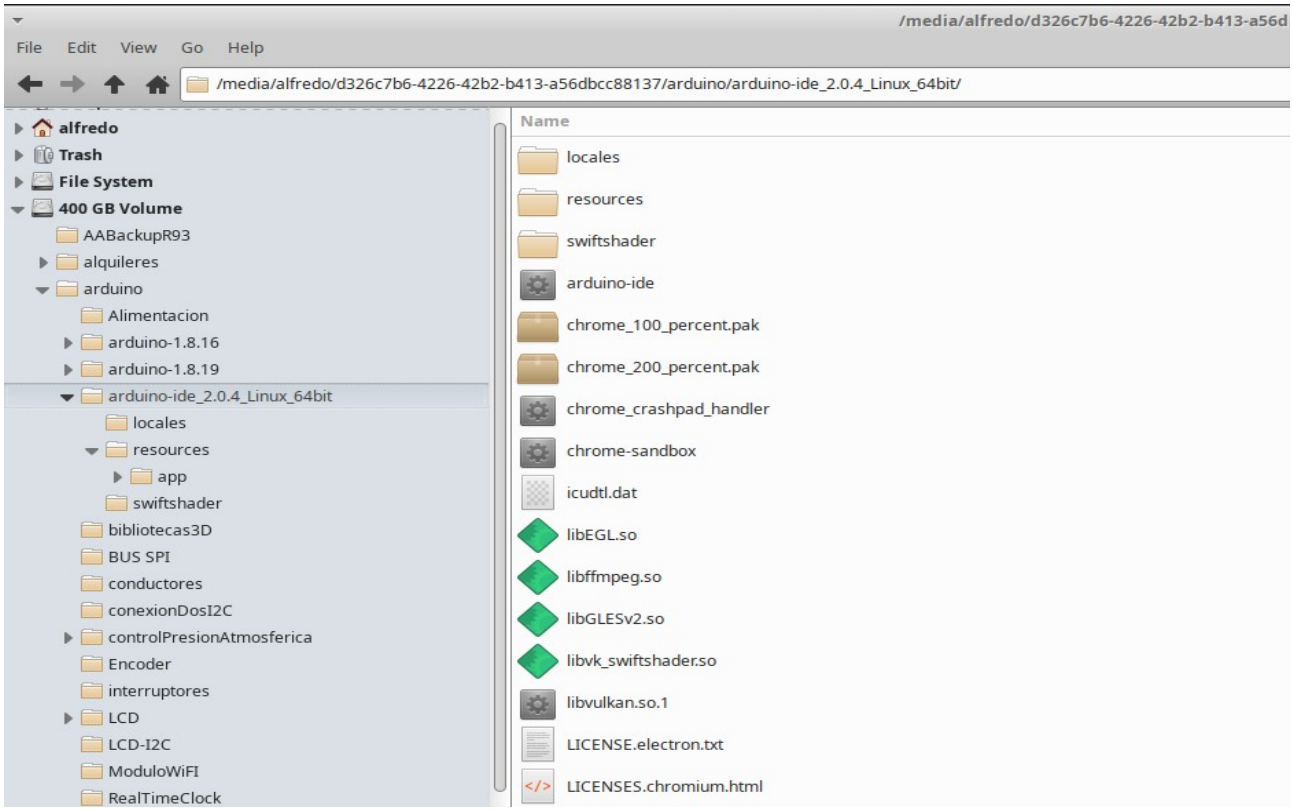


Figura 4

Al hacer doble clic sobre el archivo arduino-ide, abrirá la ventana mostrada en la Figura 5, que es el entorno de programación.

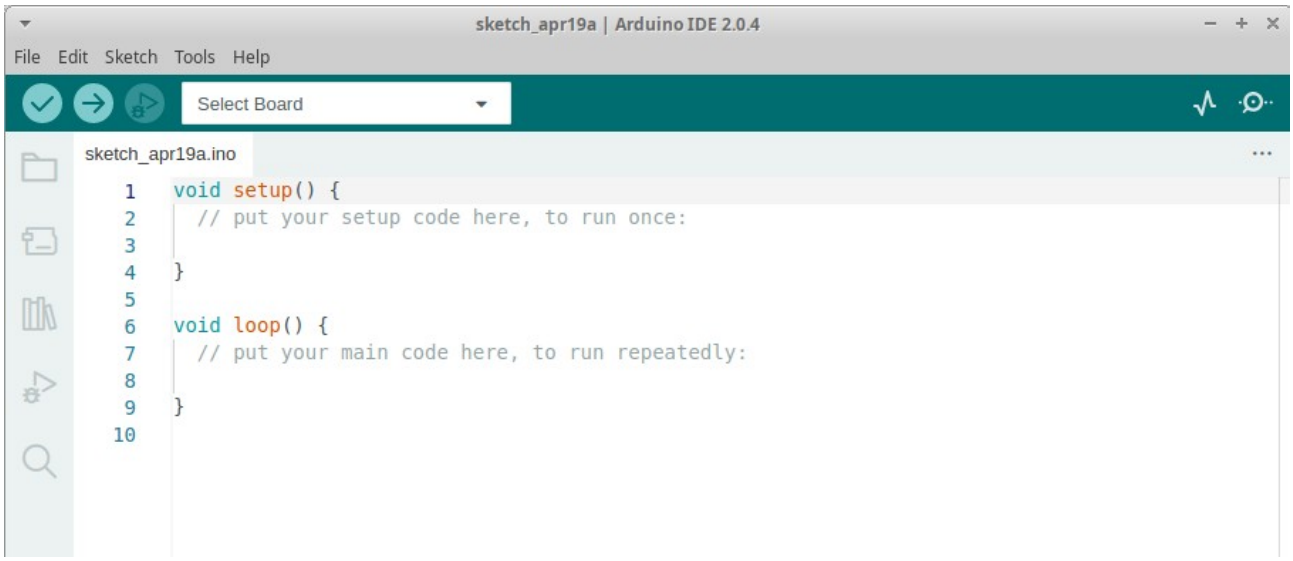


Figura 5

Si lo desea puede crear un lanzador en el escritorio, de manera que tenga un icono para ejecutar sus trabajos.

Instalación en Windows

Descarge la versión que corresponde según se indica en la Figura 6. No tenemos experiencia con la instalación en Windows, pero hallará todos los detalles en las siguiente dirección https://docs.arduino.cc/software/ide-v1/tutorials/Windows

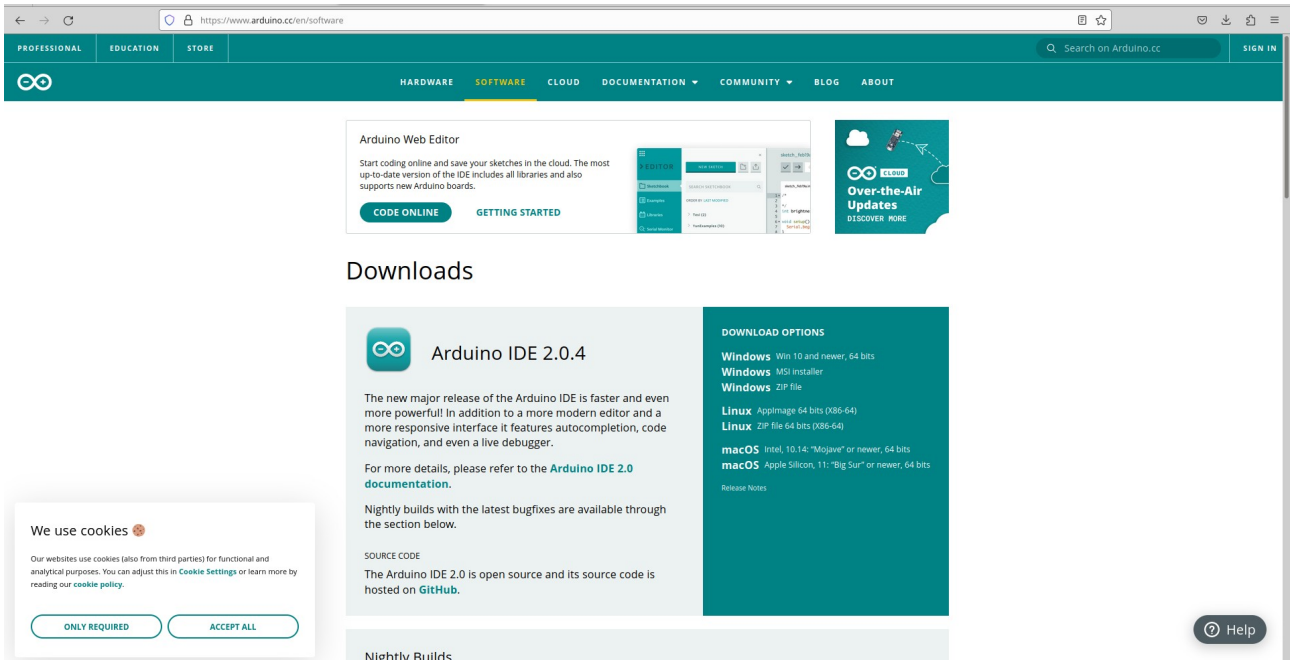


Figura 6

Conexión de Arduino a la PC y sensor DHT11

Conexión de Arduino a PC

1- Inserte el puerto USB B (Figura 7) del cable en el puerto correspondiente de la placa Arduino UNO.

Figura 7

El puerto USB A (Figura 8) del cable insértelo en un puerto USB de la computadora.

Figura 8

Se debe encender una luz, en realidad un LED (light emitting diode) que se indica con ON.
2- desenchufe el puerto USB A.

Conexión del sensor DHT11

El sensor DHT11 tiene tres pines (Figura 9)

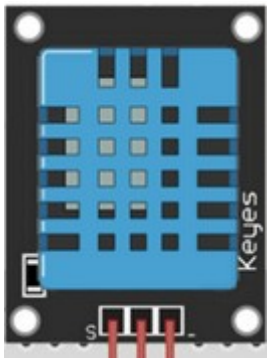


Figura 9

Si miramos el DHT11 como muestra la Figura 9, los pines de izquierda a derecha son: VCC, señal y GND
VCC, es el pin de alimentación que se conecta a 5 Volt del arduino
El pin señal, es por donde fluye la información del sensor hacia el Arduino
GND, es la tierra o neutro y se conecta tambien a la placa Arduino.
Coloque en cada pin un cable a través del conector hembra, Figura 10

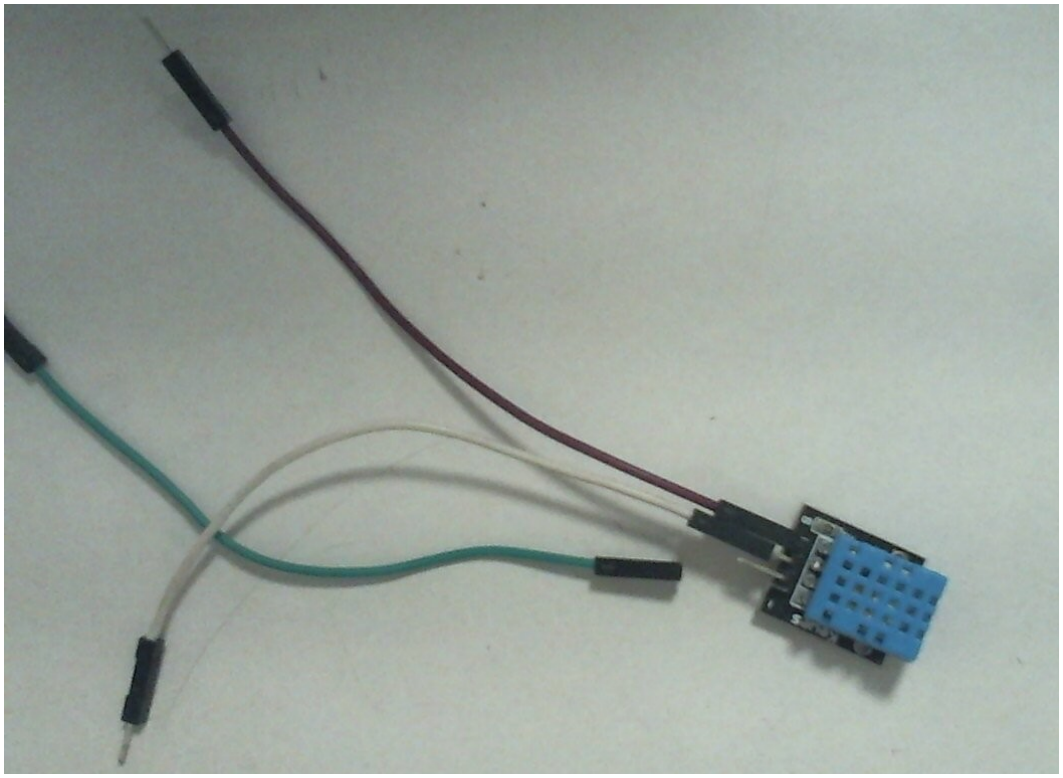


Figura 10

En la Figura 10, el cable marrón corresponde a VCC, el blanco a señal y el verde que en la figura aun está desconectado corresponde a GND.

Conexión del DHT11 a Arduino

La placa arduino UNO tiene varios puertos hembras que se hallan a ambos lados de la placa. Los puede identificar en la Figura 11. De estos utilizaremos en este caso solo 5V, GND y 7. Estos puertos son del tipo hembra por lo que allí deberá introducir los pines machos de los cables que ya se hallan unidos al DHT11.

Conecte los cables del DHT11 de la siguiente manera

DHT11	ARDUINO UNO
pin 1: VCC	5V
pin 2: señal	puerto 7
pin 3: GND	GND (hay tres posibles)

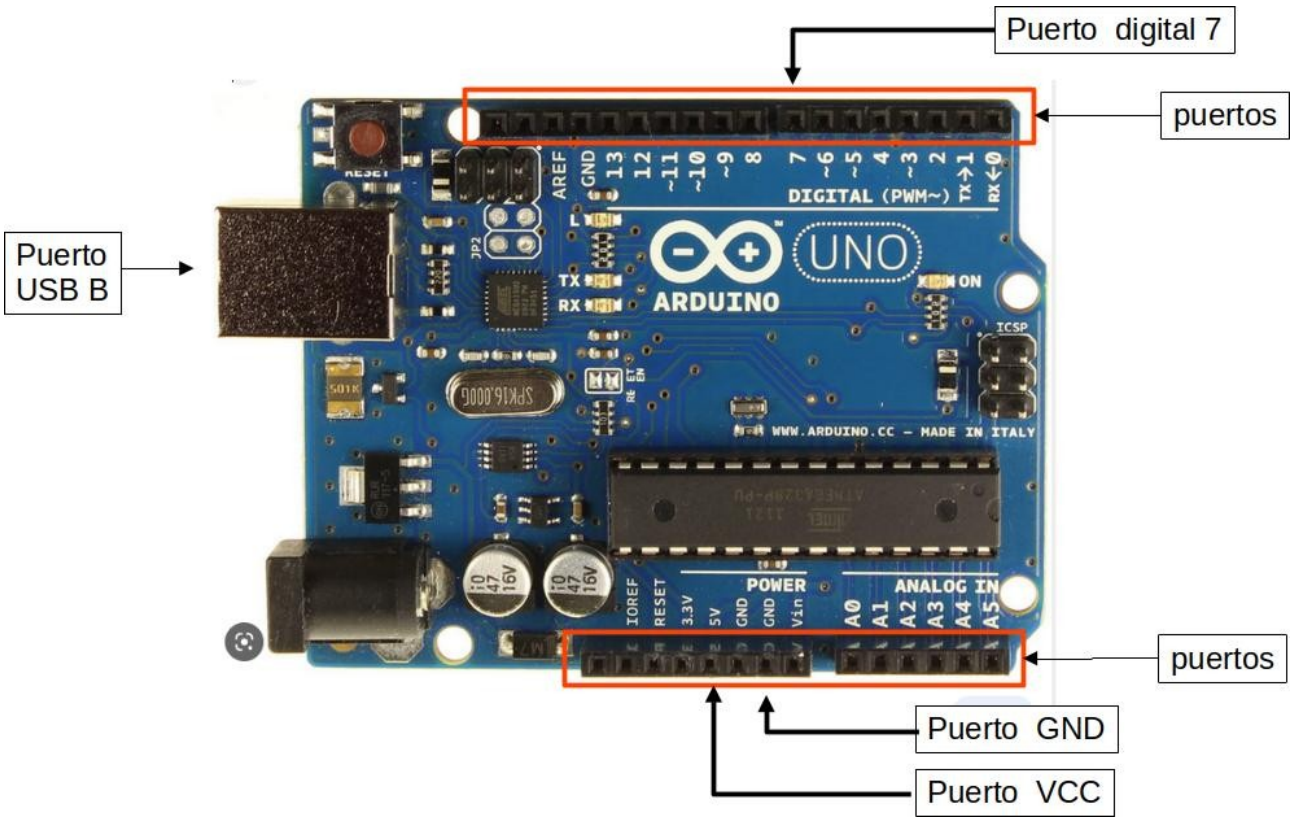


Figura 11

La conexión debería quedar como muestra la Figura 12

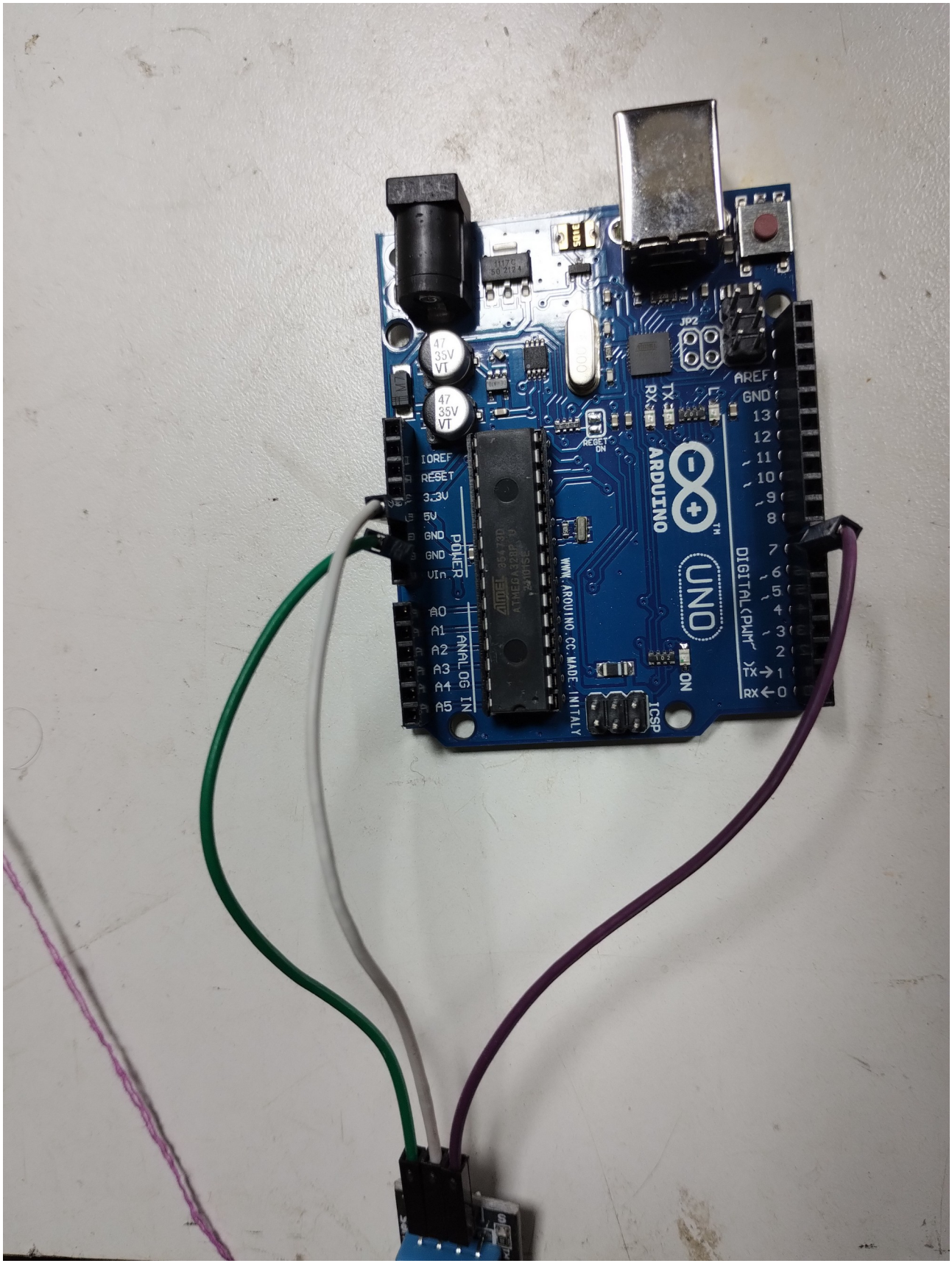

47
35V
VT
47
35V
VT
JP2
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL(PWM~)
TX
RX
RESET
ON
IOREF
RESET
3.3V
5V
GND
GND
VIn
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
ATMEGA328P-U
ARDUINO
UNO
WWW.ARDUINO.CC.MADE.INITALY
ON
ICSP

Figura 12

Software para control del DHT11

Todos los sensores y equipos que podemos utilizar con Arduino, al igual que las funciones de R que hemos venido utilizando, pertenecen a bibliotecas y ya han sido desarrolladas por otros, pudiendo hallar diferentes bibliotecas y funciones. Éstas ya han sido probadas y modificadas por muchos desarrolladores con más experiencia que la que tenemos en este momento. Por lo tanto seguimos con la misma filosofía con que trabajamos en R. Los pasos serían

1- Tenemos la necesidad o curiosidad de medir algo. Entonces buscamos un sensor que se haya desarrollado para Arduino. Esto es sencillo en un buscador de internet colocamos palabras como "medir presión atmósférica Arduino" y tendremos seguramente varios desarrollos, con todas las explicaciones, en formato de texto o en vídeo.

2- Elegimos uno de estos desarrollos y allí hallaremos detalles del sensor, algunos detalles ultratécnicos, en principio podemos obviarlos. Si es necesario conocer del sensor lo que se conoce como "pinout" que no es otra cosa que la referencia cruzada entre el número de cada pin del sensor y su función, como explicamos más arriba en una tabla para el DHT11.

3- También necesitamos como se relaciona el pinout del sensor con el Arduino. Esto en general lo hallará de manera esquemática como muestra la Figura 13, donde no nos queda duda de donde se coneccta cada pin del DHT11. Notará en la figura que el pin 1 está conectado al pin 2 de ARduino, cuando nuestro desarrollo fue al pin 7. Esto es posible, tenemos 13 puertos de entrada para la señal, que serán utilizados según el número de sensores que tengamos y los requerimientos de los mismos. Además puede ocurrir que algun sensor DHT11 tenga otro pinout. Como en este caso que es señal, VCC y GND, en lugar del orden que tiene nuestro sensor: VCC, señal, GND. En general no se producen daños a los componentes si se hace una conexión errónea. Lo que si, el sistema no funcionará.

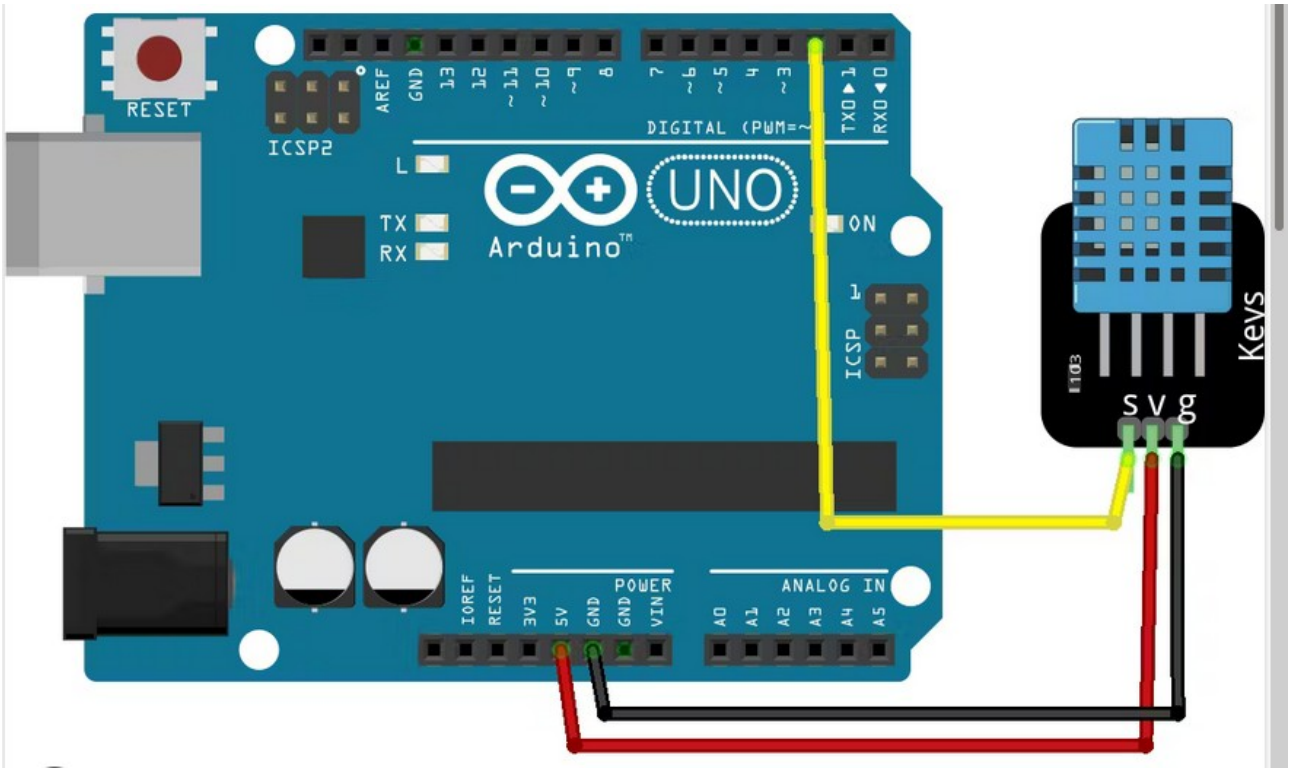


Figura 13

también puede hallar un dibujo o foto en el sensor se halla montado en lo que se llama un protoboard, que es un dispositivo que da más robustez al montaje de prueba. En estos montajes los cables en lugar de ir al sensor, van al protobard que internamente conecta con el sensor, Figura 14. El protoboard es la plancha blanca poliperforada, que actuan a manera de pines hembra. En este caso necesitaríamos cables con ambos extremos macho.

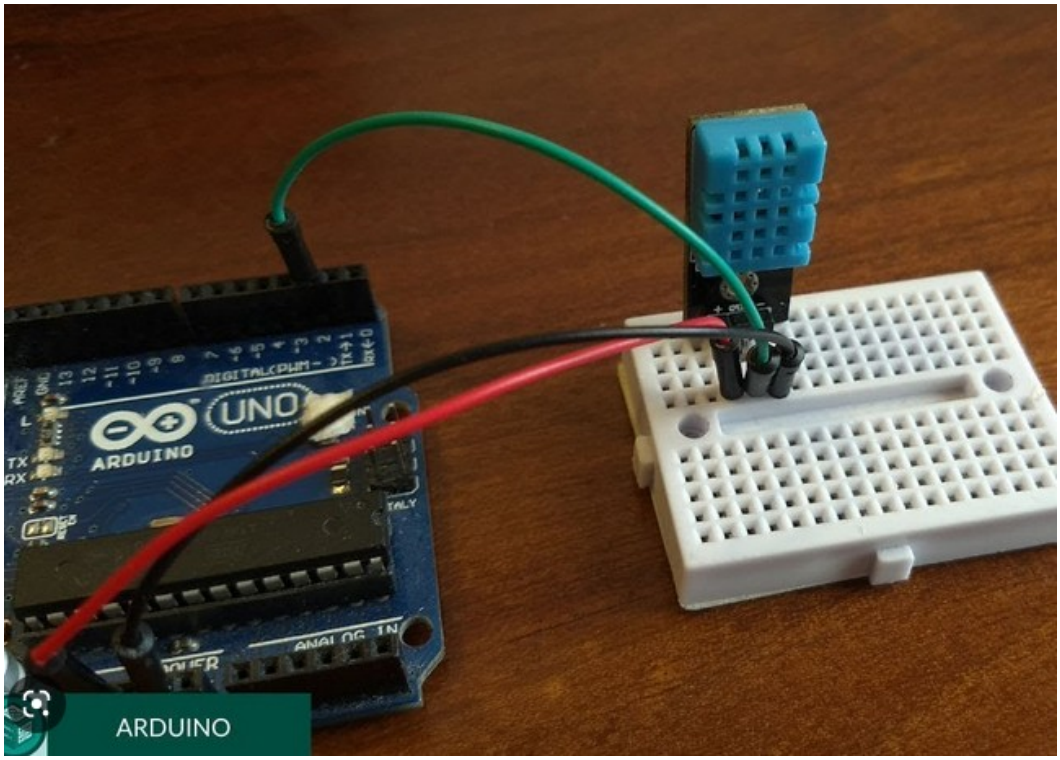


Figura 14

4- Una vez armado el circuito necesitamos un software, que también encontraremos en las páginas de internet. Pueden ser ligeramente distintos, pero todos en general funcionan muy bien. En caso que no funcione uno, elija otro. No será parte de este curso desarrollar el software, ya que se realiza en lenguaje C, que con el tiempo irá viendo que es muy parecido al utilizado para desarrollar un script en R.

Realizamos una búsqueda en un buscador de internet cruzando algunas palabras como DHT11 arduino sensor temperatura. Hallaremos varios sitios. Elegimos el link que se da a continuación

https://programarfacil.com/blog/arduino-blog/sensor-dht11-temperatura-humedad-arduino/

Hallará en la página detalles e información que seguramente ya algo de ella comprenderá, aunque nunca haya estudiado una palabra de electrónica. Imaginese lo que será dentro de 1 año!!! Si bajamos en la página hallaremos la biblioteca necesaria para poder utilizar el sensor Figura 15. En el texto verá resaltado proporciona Adafruit. Haciendo clic en el link nos llevará a la biblioteca

Figura 15

Al hacer clic en "proporciona Adafruit" se abrirá una página como muestra la Figura 16

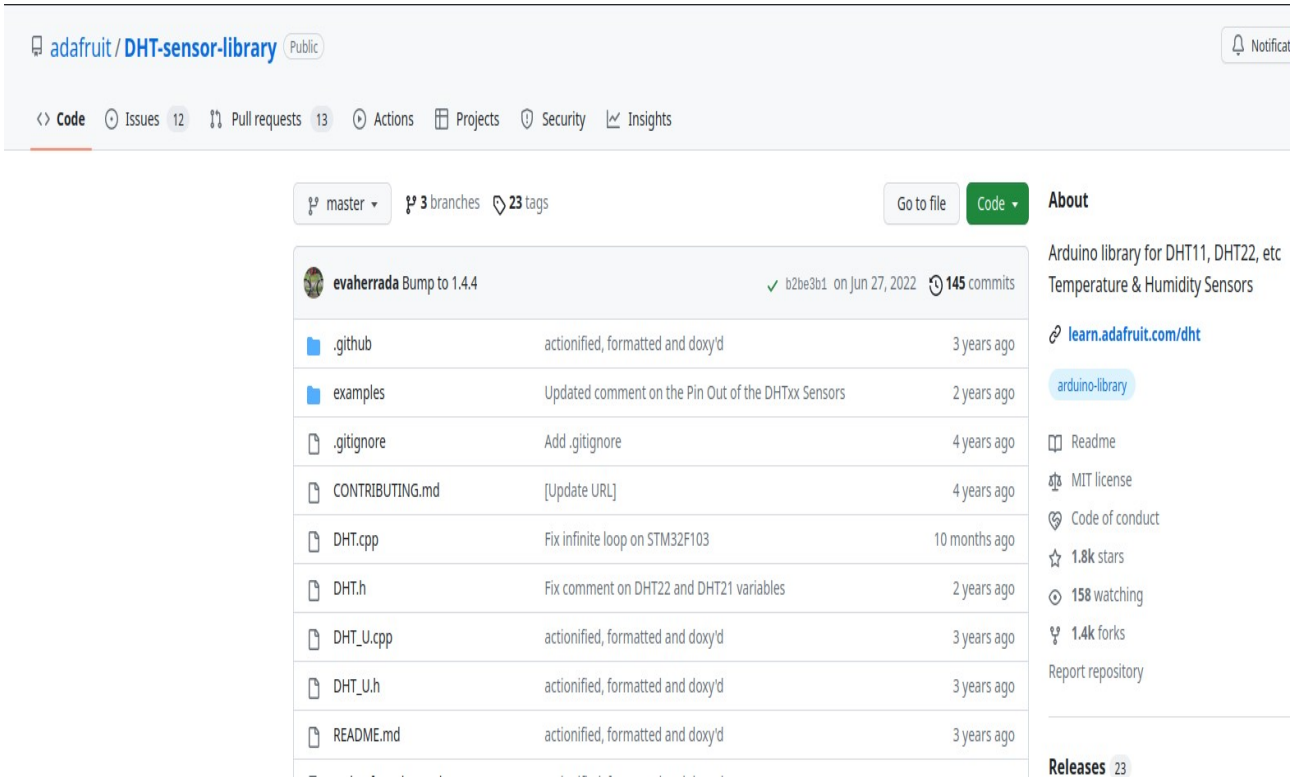


Figura 16

Haga clic en Code (botón verde de la Figura 16), desplegará una ventana como muestra la Figura 17

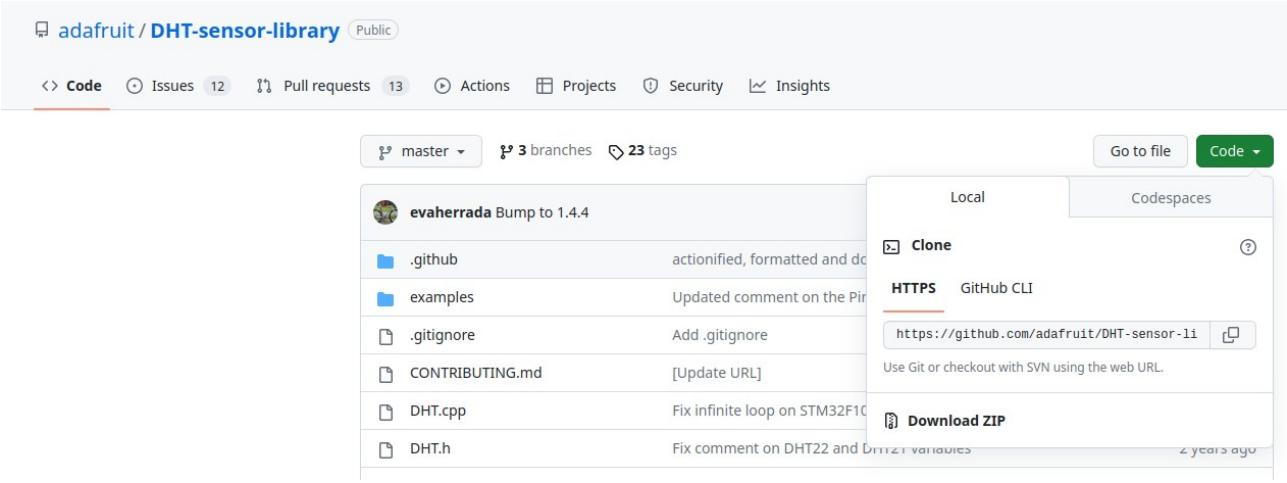


Figura 17

haga clic en "Download.ZIP", esto comenzará la descarga del archivo que irá a la carpeta que tenga configurada, habitualmente la carpeta Descagas o Downloads.
Si vamos luego a esa carpeta hallaremos el archivo que tiene el nombre DHT-sensor-library-master.zip. Más adelante utilizaremos esta biblioteca.

Seguimos ahora con el sketch o software que enviaremos a Arduino UNO para que cumpla la función de medir temperatura y humedad.
Abrimos la interfaz de desarrollo haciendo doble clic sobre el icono de arduino que hemos instalado. Se debe abrir la interfaz de desarrollo, Figura 18.

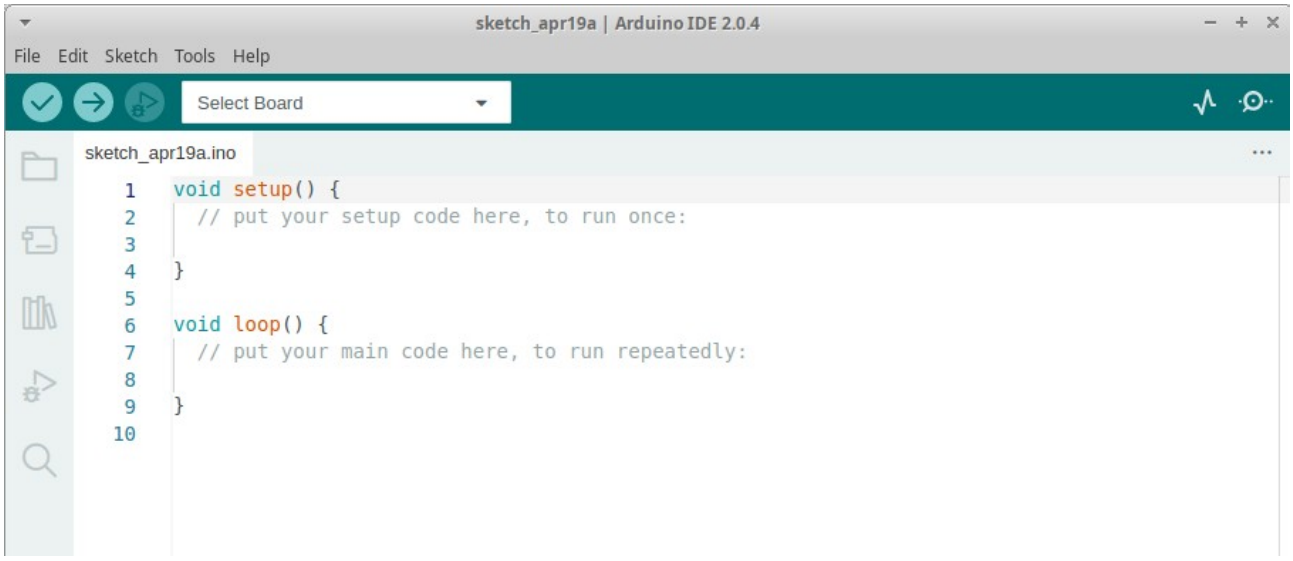


Figura 18

Si volvemos a la página web donde buscamos la biblioteca, más abajo hallaremos es sketch, divido en tres partes en este caso, Figura 19.

Figura 19

la primer parte titulada: Libreria DHT y variables la copiaremos en la interfaz de desarrollo, antes de la línea void setup().
La segunda parte titulada Función setup, la copiaremos en la interfaz de desarrollo, entre void setup() y void loop()
La tercer parte titulada Función loop, la copiaremos debajo de la línea void loop()
La Figura 20 esquematiza este proceso

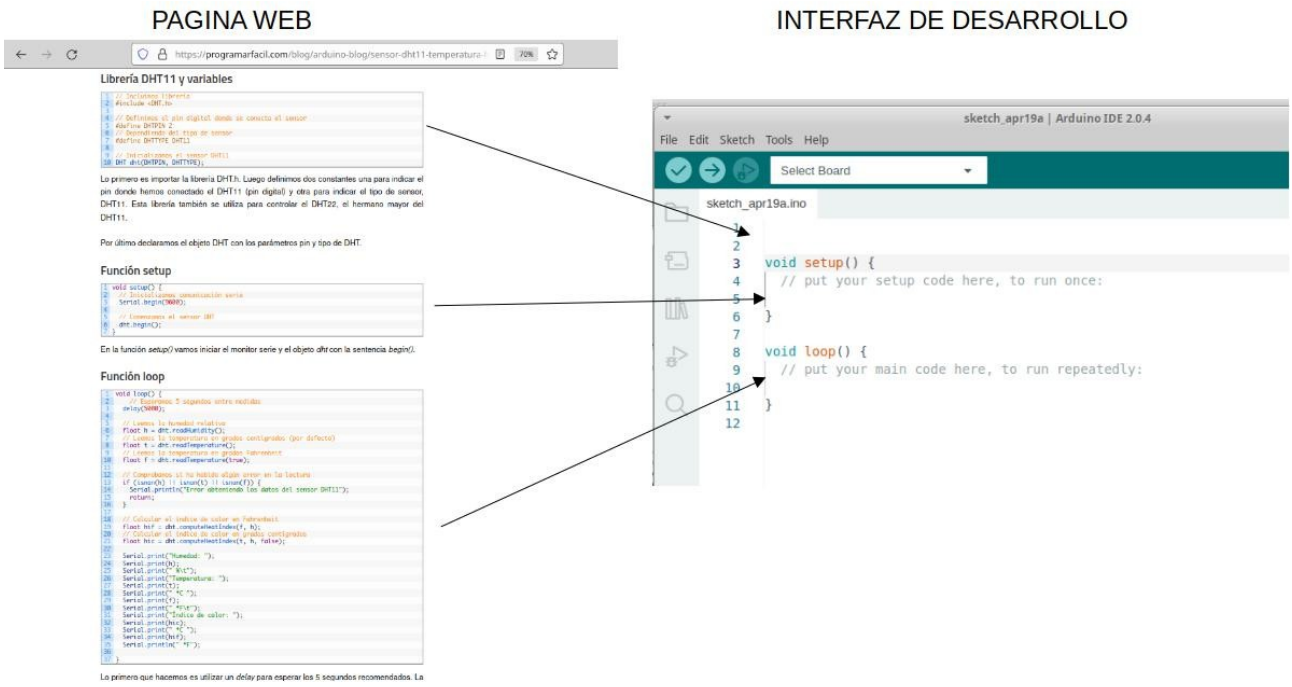


Figura 20

Luego del proceso la interfaz quedará como muestra la Figura 23. En la figura no se alcanza a ver todo el software, por cuestiones de espacio. Como verá en la línea 6 del sketch se realizó un cambio, en lugar de 2 se ha colocado 7, que es el puerto que hemos utilizado para la conexión del sensor.

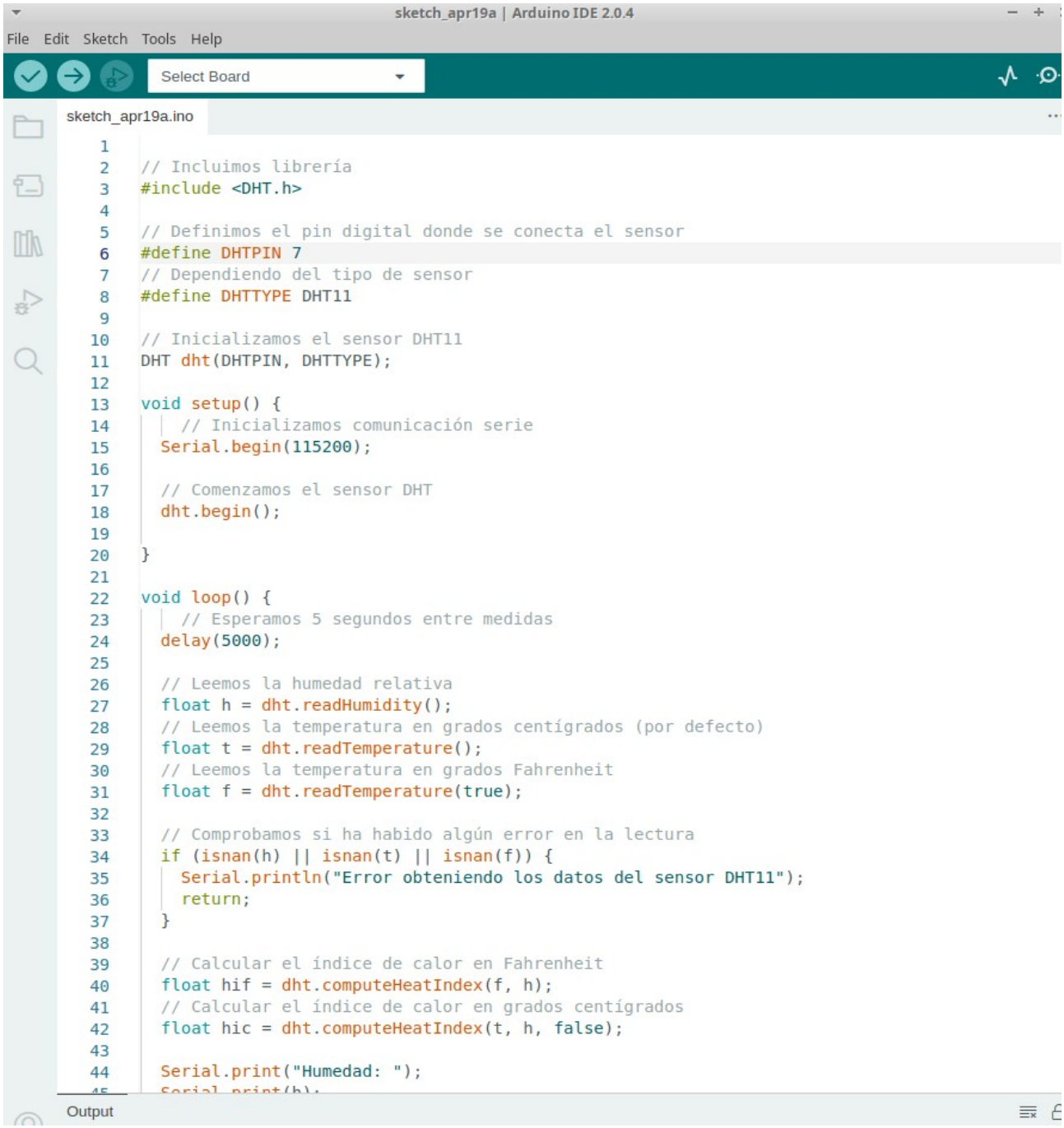


Figura 21

Nos faltan pocas configuraciones. En la interfaz de desarrollo desplegamos el menú
Tools > Boards > Arduino AVR Boards
que desplegará un listado de placas arduino, hacemos clic en Arduino uno, Figura 22.

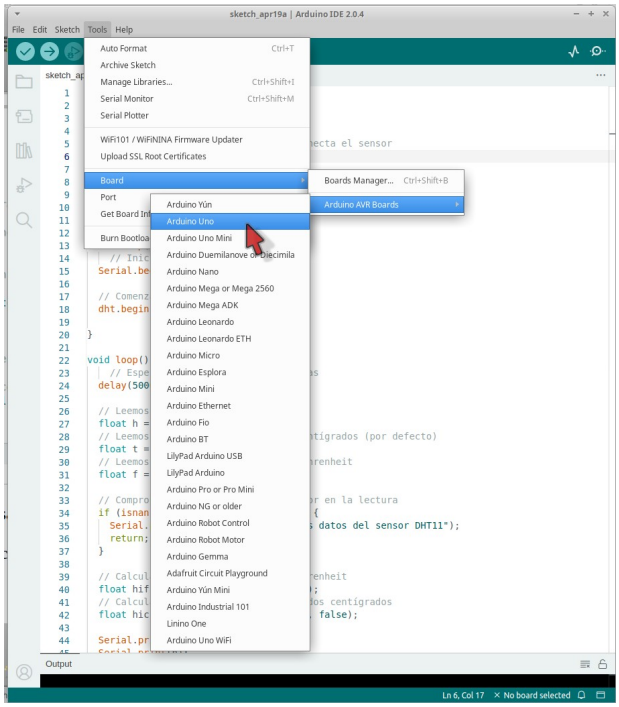


Figura 22

El último paso de configuración consiste en elegir el puerto serial. Desplegamos el menu
Tools > Ports
y elegimos el puerto que nos muestra

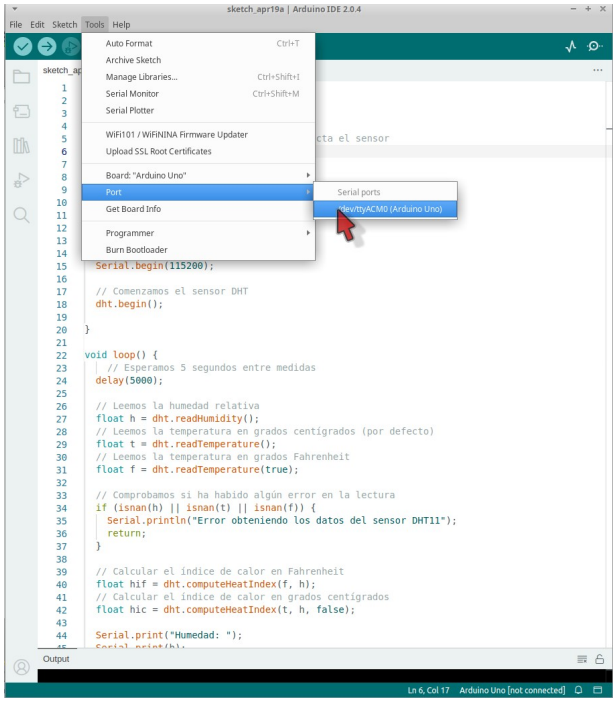


Figura 23

Ahora tenemos que comprobar si todo esta para funcionar. Del menú ejecutamos
Sketch > Verify/Compile
En ese paso verifica si todo está orden para funcionar. Si es así luego de ejecutar la acción nos mostrará debajo del sketch una ventana donde aparece un texto en blanco que nos informa sobre

cuestiones de memoria ocupada y disponible. Si no aparece nada en rojo, estamos listos!!!! Figura 24

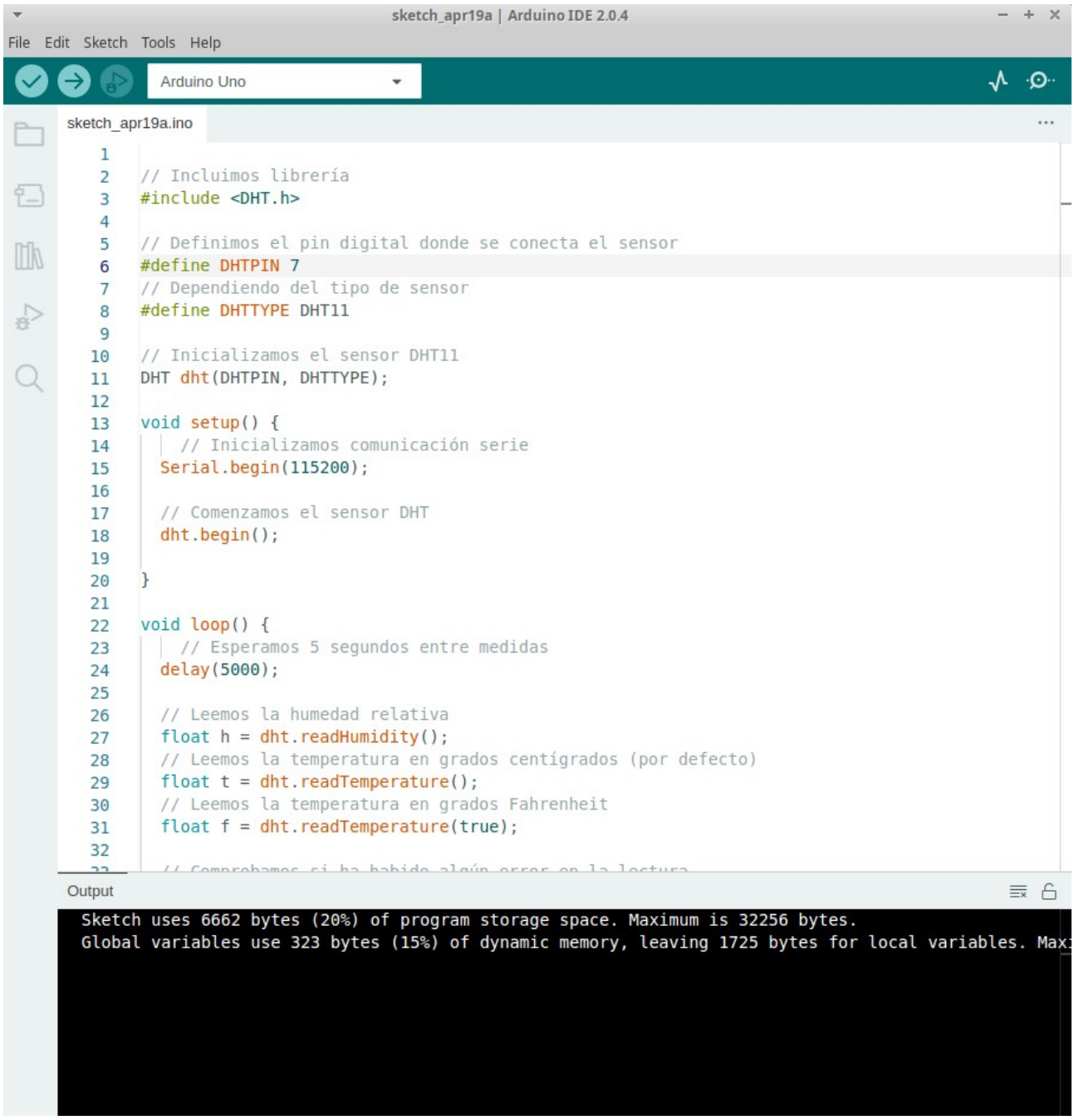


Figura 24

Solo nos resta cargar el sketch en la placa arduino. Para ello desplegamos el menu
Sketch > Upload
La misma acción se logra oprimiento el botón que tiene una flecha hacia la derecha en la barra de herramientas, Figura 25.

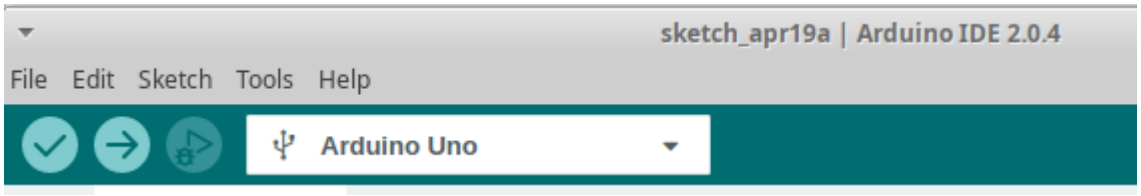


Figura 25

Si luego de esta acción en la ventana inferior ha quedado el mensaje escrito en blanco y nada en rojo o palabras Error, Arduino está leyendo el sensor!!!!
Para ver las lecturas que está haciendo la información que está enviando por el puerto serial, oprimimos en la barra de herramientas el icono de monitor serial, que es el último icono a la derecha de la barra ver

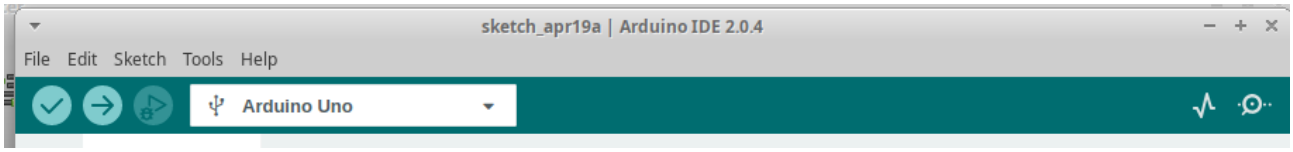


Figura 26

Al oprimir el icono  , se abrirá el monitor serial en la parte inferior mostrando las mediciones de temperatura y humedad, Figura 27. En la figura se muestra una situación en que el sensor está dando un error.

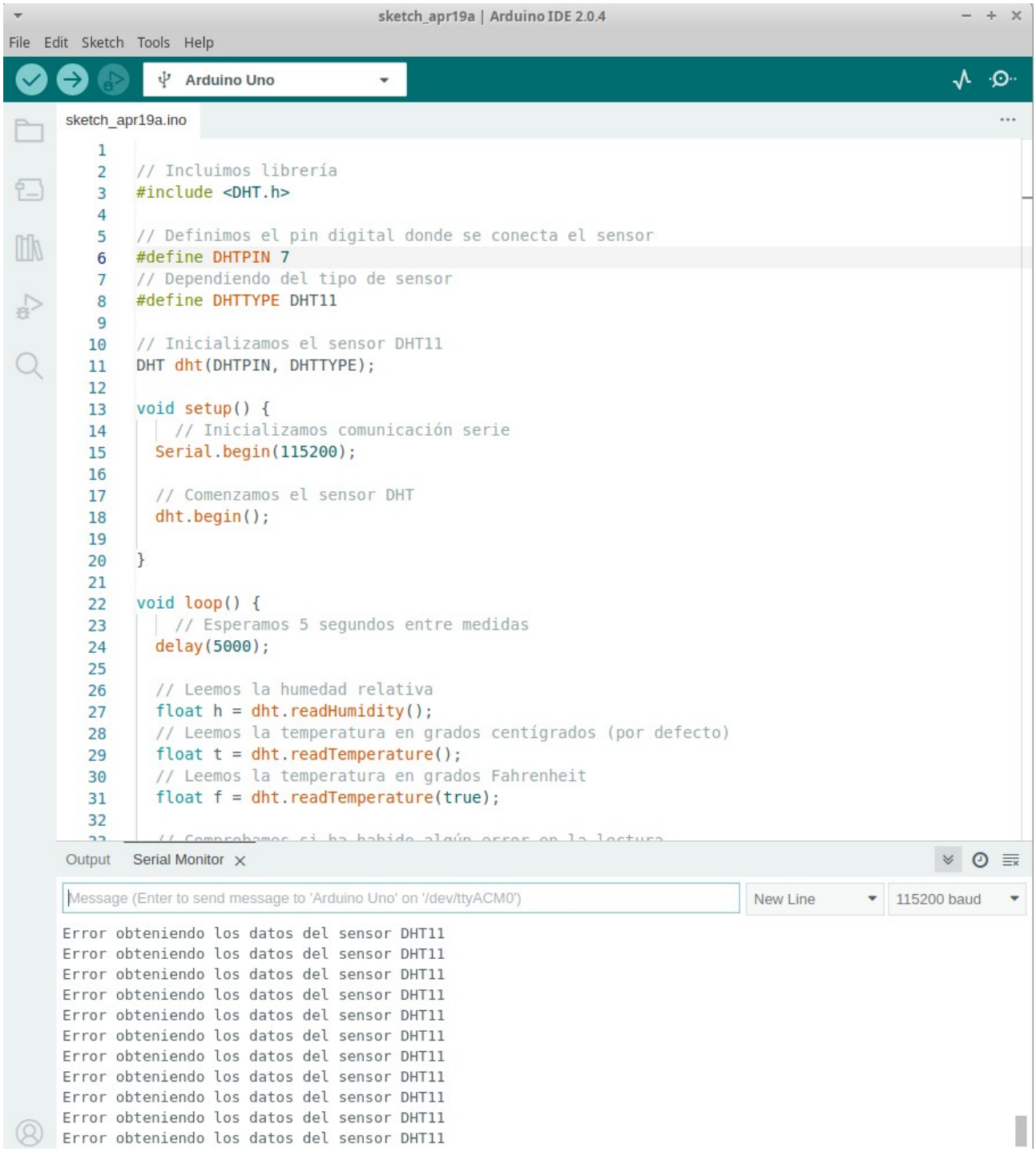


Figura 27

En caso que funcione correctamente se observará la salida que muestra la

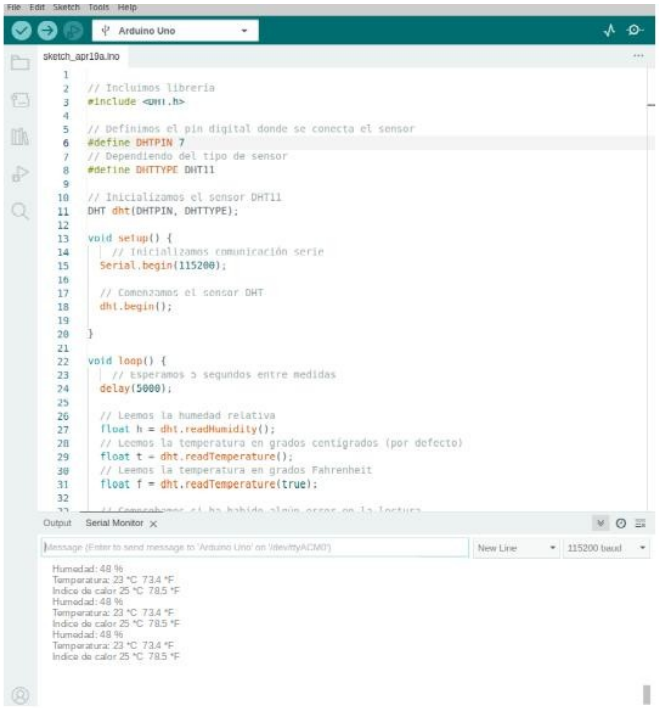


Figura 28

Finalmente guardaremos el sketch, en este caso con el nombre R97.ino. Esto ya es sencillo, del menú File elegimos
File > Save as
Como en todo procedimeinto Save as... colocamos el nombre R97, elegimos la carpeta que deseamos. Arduino agrega automaticamente la extensión .ino.

Luego de este arduo camino estamos en condiciones de recibir los datos de R. La tarea no será menos ardua que la de esta clase. Como siempre el primer paso es el de mayor dificultoso. Solo bastará sentarse a descansar y pensar en los 8 módulos pasados!!!
Los espero en R98!!

Módulo 9 – clase 8

Comunicación Arduino - R

Si está leyendo esta clase es porque sorteo su primer contacto con Arduino o tiene intenciones de terminar haciéndolo.
En clases anteriores comenzamos a tomar contacto con componentes electrónicos capaces de cambiar su señal de salida electrónica en respuesta a una variable. En la clase R97 trabajamos con el sensor DHT11, que permite medir temperatura y humedad. El DHT11 genera una señal digital que envía dicha información por un pin de salida. Si lo tenemos conectado a un Arduino y éste a una computadora a través de un cable USB A/B, podemos introducir dicha medida en nuestra computadora.
Modificación sketch para lectura del DHT11

Por supuesto, además necesitamos un sketch en Arduino que capte la señal del DHT11 y la redirija al puerto serial. Analicemos nuevamente el sketch de la clase anterior, indicando en colores diferentes a cada sector del mismo. A la derecha se hace una breve explicación. Recuerde que las líneas que comienzan con // son comentarios y las líneas de instrucciones terminan con ";". Se

muestran tres areas con diferente fondo, separando así la zona de declaraciones de variables, setup y loop

sketch	explicación
// Incluimos librería #include <DHT.h>	inclusión de biblioteca para manejo de DHT11

// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 7
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11
// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);
	definición del puerto de arduino por el cual ingresa la señal, este caso es el puerto digital número 7
	elección del tipo de sensor a utilizar. La biblioteca DHT.h puede manejar el DHT11 y DHT22
	inicialización del DHT11. Entre paréntesis se indica puerto y tipo de sensor

void setup() {
// Inicializamos comunicación serie
Serial.begin(115200);
 	
// Comenzamos el sensor DHT
dht.begin();
 	
}
	inicio de la comunicación serial entre arduino y PC a 115200 baudios
	
	inicio de funcionamiento del DHT11
	
	llave que cierra la instrucción void setup()

void loop() {
 	
// Esperamos 5 segundos entre medidas
delay(5000);
 	
// Leemos la humedad relativa
float h = dht.readHumidity();
 	
// Leemos la temperatura en grados centígrados (por defecto)
float t = dht.readTemperature();
 	
// Comprobamos si ha habido algún error en la lectura
if (isnan(h) || isnan(t)) {
Serial.println("Error obteniendo los datos del sensor DHT11");
return;
}
 	
// Calcular el índice de calor en grados centígrados
float hic = dht.computeHeatIndex(t, h, false);
 	
Serial.print("Humedad: ");
Serial.print(h);
Serial.print(" %\t");
 	
Serial.print("Temperatura: ");
Serial.print(t);
Serial.print(" *C ");
 	
Serial.print("Índice de calor: ");
Serial.print(hic);
Serial.print(" *C ");
 	
}
	Las instrucciones que se hallan luego de void loop() se repetiran cada 5 segundo
	
	tiempo de espera (5 segundos) entre medición
	
	DHT11 lee la humedad y la asigna a una variable llamada h
	
	DHT11 lle la temperatura y la asigna a la variable t
	
	evalua si la variable h o t no ha sido medida, en tal caso no tendrá valor e informará del error con el mensaje Error obteniendo los datos del sensor DHT11
	
	calcula con las variables medidas: h y t, el indice de caor y lo asigna a la variable hic
	
	Imprime en el monitor serial el mensaje:
	Humedad: valor de la variable h %
	ejemplo: Humedad: 45 %
	
	Imprime en el monitor serial el mensaje:
	Temperatura: valor de la variable t *C
	ejemplo: Temperatura: 35 *C
	
	Imprime en el monitor serial el mensaje:
	Índice de calor: valor de la variable hic *C
	ejemplo: Índice de calor: 35 *C
	
	llave que cierra el loop y repite todo el proceso luego de 5 segundos.

Modificaremos ahora el script para que envie por el puerto serial la misma información, pero más restringida. Esta restricción nos permitirá luego desde R manejar dicha información con más facilidad.
Eliminaremos del sketch la variable hic y además eliminaremos de las funciones Serial.print, los detalles que indican la variable y su unidad. El sketch quedará como se muestra en la columna 2 de la tabla siguiente. A la izquierda tiene el script original para comparar los mínimos cambios. Las líneas en rojo y tachadas son las que se eliminarán y las en color rojo, las que sufren modificaciones

sketch original	sketch modificado. Se muestran tachadas y en color rojo las líneas que se eliminan y en color rojo las modificadas o agreadas

// Incluimos librería
#include <DHT.h>
 	
// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 7
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11
// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);
 	
void setup() {
// Inicializamos comunicación serie
Serial.begin(115200);
 	
// Comenzamos el sensor DHT
dht.begin();
 	
}
 	
void loop() {
 	
// Esperamos 5 segundos entre medidas
delay(5000);
 	
// Leemos la humedad relativa
float h = dht.readHumidity();
 	
// Leemos la temperatura en grados centígrados (por defecto)
float t = dht.readTemperature();
 	
// Comprobamos si ha habido algún error en la lectura
if (isnan(h) || isnan(t)) {
Serial.println("Error obteniendo los datos del sensor DHT11");
return;
}
 	
// Calcular el índice de calor en grados centígrados
float hic = dht.computeHeatIndex(t, h, false);
 	
Serial.print("Humedad: ");
Serial.print(h);
 	
Serial.print(" %\t");
 	
Serial.print("Temperatura: ");
Serial.print(t);
	// Incluimos librería
	#include <DHT.h>
	
	// Definimos el pin digital donde se conecta el sensor
	#define DHTPIN 7
	// Dependiendo del tipo de sensor
	#define DHTTYPE DHT11
	// Inicializamos el sensor DHT11
	DHT dht(DHTPIN, DHTTYPE);
	
	void setup() {
	// Inicializamos comunicación serie
	Serial.begin(115200);
	
	// Comenzamos el sensor DHT
	dht.begin();
	
	}
	
	void loop() {
	
	// Esperamos 5 segundos entre medidas
	delay(5000);
	
	// Leemos la humedad relativa
	float h = dht.readHumidity();
	
	// Leemos la temperatura en grados centígrados (por defecto)
	float t = dht.readTemperature();
	
	// Comprobamos si ha habido algún error en la lectura
	if (isnan(h) || isnan(t)) {
	Serial.println("Error obteniendo los datos del sensor DHT11");
	return;
	}
	
	~~// Calcular el índice de calor en grados centígrados~~
	~~float hic = dht.computeHeatIndex(t, h, false);~~
	
	~~Serial.print("Humedad: ");~~
	Serial.print(h);
	Serial.print(",") ; // coloca una coma luego del valor de h
	~~Serial.print(" %\t");~~
	
	~~Serial.print("Temperatura: ");~~
	Serial.println(t);

Serial.print(" *C "); Serial.print("Índice de calor: "); Serial.print(hic); Serial.print(" *C "); }	~~Serial.print(" *C ");~~ ~~Serial.print("Índice de calor: ");~~ ~~Serial.print(hic);~~ ~~Serial.print(" *C ");~~ }

Como se puede observar se eliminaron las sentencias que escribian Humedad y su unidad, Temperatura y su unidad. Además se cambió Serial.print(t) por Serial.println(t), que imprimirá la temperatura en el monitor serial y dará un retorno de carro, dejando preparada para que la próxima medición pase a la línea siguiente. Serial.println() es equivalente a \t, que aparece en algunas líneas del sketch original.
Entonces el sketch definitivo con el que intentaremos ingresar a la PC y de allí dirigirlo a R será el que se muestra a continuación y corresponde a la eliminación de las líneas indicadas en la segunda columna de la tabla anterior

```
// Incluimos librería
#include <DHT.h>

// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 7
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11
// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
// Inicializamos comunicación serie
Serial.begin(115200);

// Comenzamos el sensor DHT
dht.begin();
}

void loop() {

// Esperamos 5 segundos entre medidas
delay(5000);

// Leemos la humedad relativa
float h = dht.readHumidity();

// Leemos la temperatura en grados centígrados (por defecto)
float t = dht.readTemperature();

// Comprobamos si ha habido algún error en la lectura
if (isnan(h) || isnan(t) ) {
Serial.println("Error obteniendo los datos del sensor DHT11");
return;
}
Serial.print(h);
Serial.print(",") ;   // coloca una coma luego del valor de h
Serial.println(t);

}
```

Modifique el sketch anterior. Para ello copie el código de la tabla anterior y reemplace todo el contenido del sketch R97.ino, luego coloque Save as y guardelo con otro nombre, por una cuestión de orden lo llamaremos R98.ino. Recuerde que la IDE de Arduino crea una carpeta con ese nombre y coloca dentro el archivo con extensión .ino.

Luego verifique que el código esté correcto ejecutando desde el menú
Sketch > Verify/Compile
si todo está en orden en la parte inferior, en Output debe hallar algo como se muestra en la Figura 1, sin textos en color rojo.

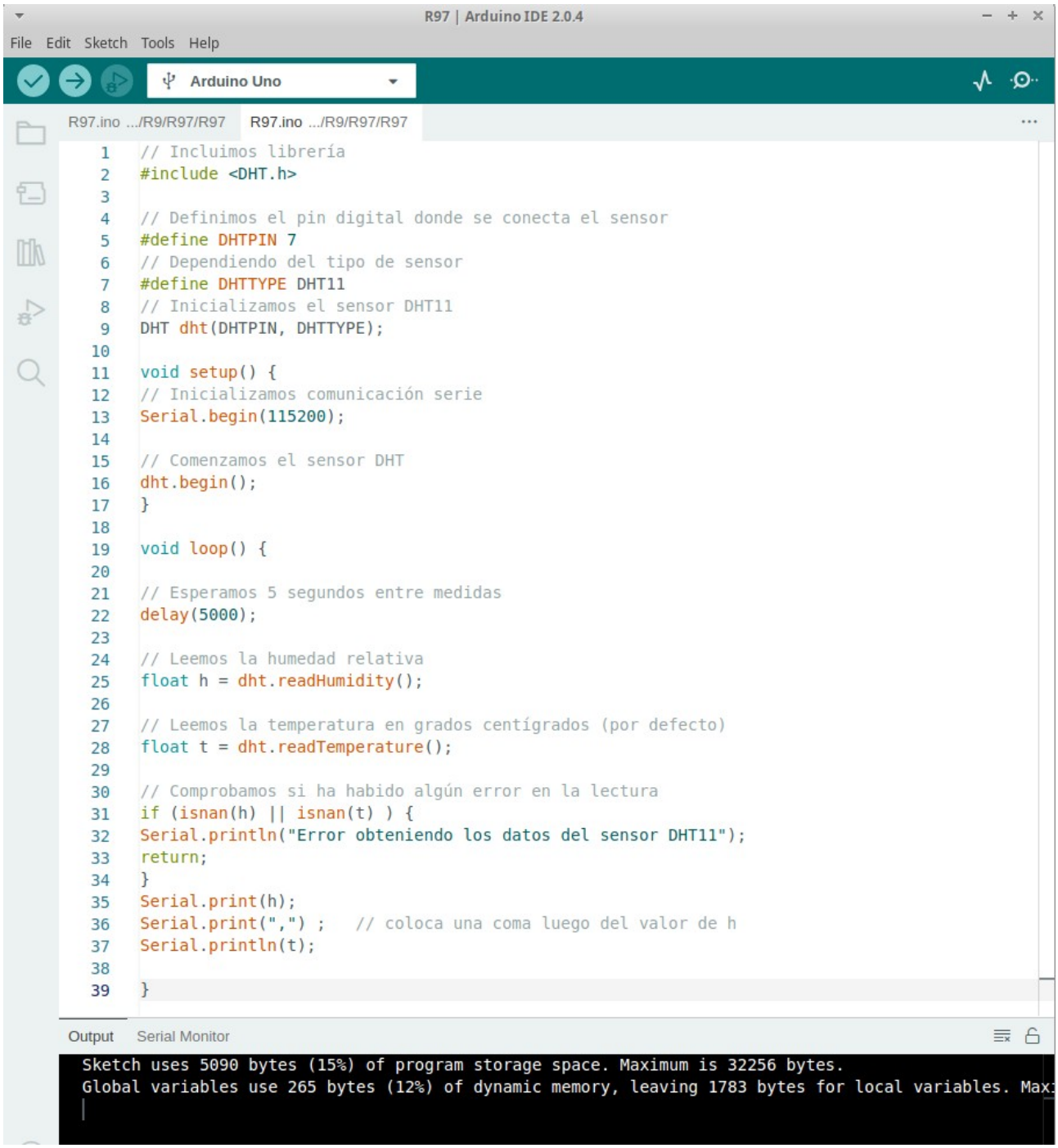


Figura 1

Luego cargue el sketch en el micropocesador, oprimiendo la flecha de la barra de herramientas ( ) o bien ejecute desde el menú
Sketch > Upload

Si no ha dado ningún error, oprima el símbolo de monitor serial
En la parte inferior se debe mostrar la medición de humedad y temperatura, separados por una coma. Cada 5 segundos debe aparecer una nueva medición, como muestra la Figura 2

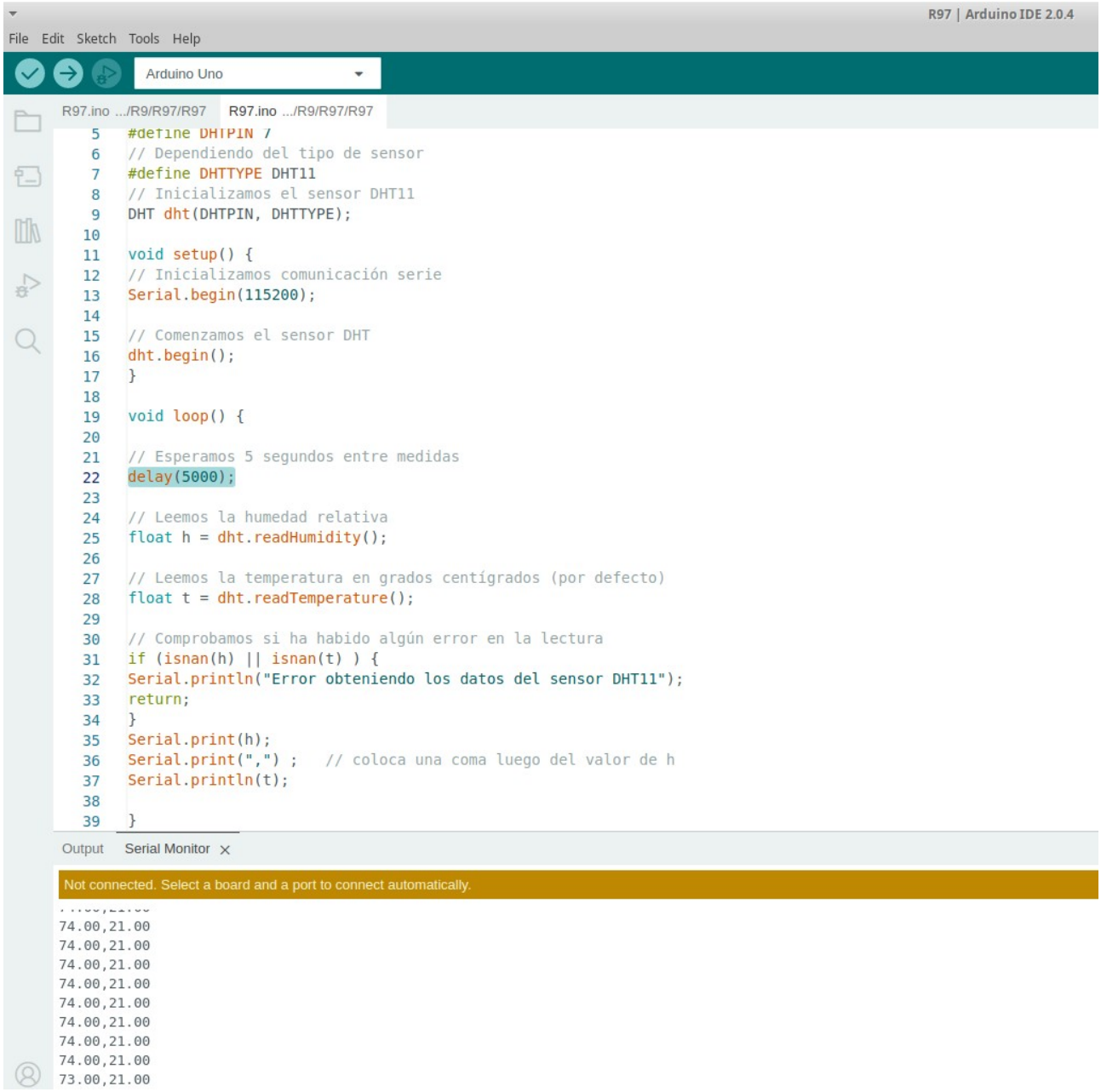


Figura 2

Ya estamos en condiciones de recibir esta información desde R!!!
Diseño del script en R para lectura del puerto serial

Para ello, el primer paso es instalar la biblioteca serial, que puede descargar desde los repositorios que ya lo viene haciendo durante el curso. Como siempre la forma más sencilla, desde la consola ejecute directamente

```
> install.packages("serial",dependencies=TRUE)
```

luego cárguela en su espacio de trabajo

```
> library(serial)
```

A continuación veremos el script que utilizaremos. Discutiremos en mayor profundidad cada línea y función utilizada en la clase siguiente. Por el momento aceptaremos el mismo, con algunas mínimas explicaciones.

script	explicacion
library(serial) #Facilita acceder al puerto de serie	carga biblioteca serial

print("Script de medición DHT11")
system("sudo chmod 777 /dev/ttyACM1") #esta línea puede eliminarse y activar el puerto manualmente de ser necesario
print("introduzca tiempo de medicion en minutos,ejemplo:30")
tiempomedicion=as.numeric(scan(file="",what="",nmax=1))
	Indica la utilidad del script
	En linux habilita acceso y lectura de puerto serial. Omitir en windows y probar
	Pide tiempo de medición. Introducir solo numero en segundos.
	Inicializa variable tiempomedicion con los segundos introducidos

#Toma tiempo de inicio
tinicial<-proc.time()[3]
	Inicializa tinicial con los segundos provistos por proc.time()

#En Port va el puerto que sugiere la IDE de arduino en el menu Tools > Ports
a<-serialConnection(port="ttyACM1",mode="115200,n,8,1",translation="lf")
	configuración de la conexión serial

#Si ya hay una conexión abierta por haber medido antes la cierra y abre nuevamente
if(isOpen(a)) close(a)
open(a)
	Cierra la conexión en caso qe estuviera abierta
	Abre la conexión

#creamos un data.frame al que llamamos th y donde guardaremos los datos de humedad y temperatura, tendrá una sola columna llamada medicion
th<-data.frame(medicion=NA)
	Creación de data.frame para almacenar datos

read.serialConnection(a)	Lee la información recibida por el puerto serial y que se halla en buffer de memoria
Sys.sleep(5) i=1	Espera 5 segundos

while(tiempomedicion*60>(proc.time()[3]-tinicial)){
#Cada medicion es una linea con los valores de humedad y temperatura
#20 bytes asegura que estoy agarrando al menos una medición entera
if(nBytesInQueue(a)>20){
	Bucle while que se realizará siempre que el tiempo transcurrido sea menor que el valor con que se inicializó la variable tiempomedición.
	Condición a tener en cuenta en función de los bytes recibidos desde el puerto serial

#Leo, guardo en variable a la que llamamos buffer
buffer<-read.serialConnection(a)
	Lee el buffer de memoria con información recibida desde DHT11 y lo asigna a la variable buffer

#buscamos en la variable buffer, los saltos los saltos de linea que ingresan por el puerto como \r
separador<-gregexpr("\r",buffer)
	Identifica con la funciòn gregexpr, la posición de los retornos de carro: \r

#nos quedamos con la parte de medición entre dos saltos de linea sucesivos, es decir, nos quedamos con una medicion y la asignamos a variable ultimo
ultimo<-substring(buffer,separador[[1]][1]+1,separador[[1]][2]-1)
	Inicializa la variable ultimo con un valor entre dos retornos de carro

#Guardo en data.frame th el valor de la variable ultimo th[i+1,1]<-ultimo print(th) i<-i+1 Sys.sleep(5) }	guarda el valor en el data.frame th muestra en pantalla el data.frame th espera 5 segundos.
}	

Copie el código de la tabla anterior, construya un archivo con el nombre R98.R.
Conecte el proyecto Arduino con el DHT11 al que previamente le cargó el sketch desarrollado, a través del cable USB con la computadora donde ejecuta R.
Luego ejecute
> source('R98.R')
Debería obtener una salida en la consola como muestra la Figura 3

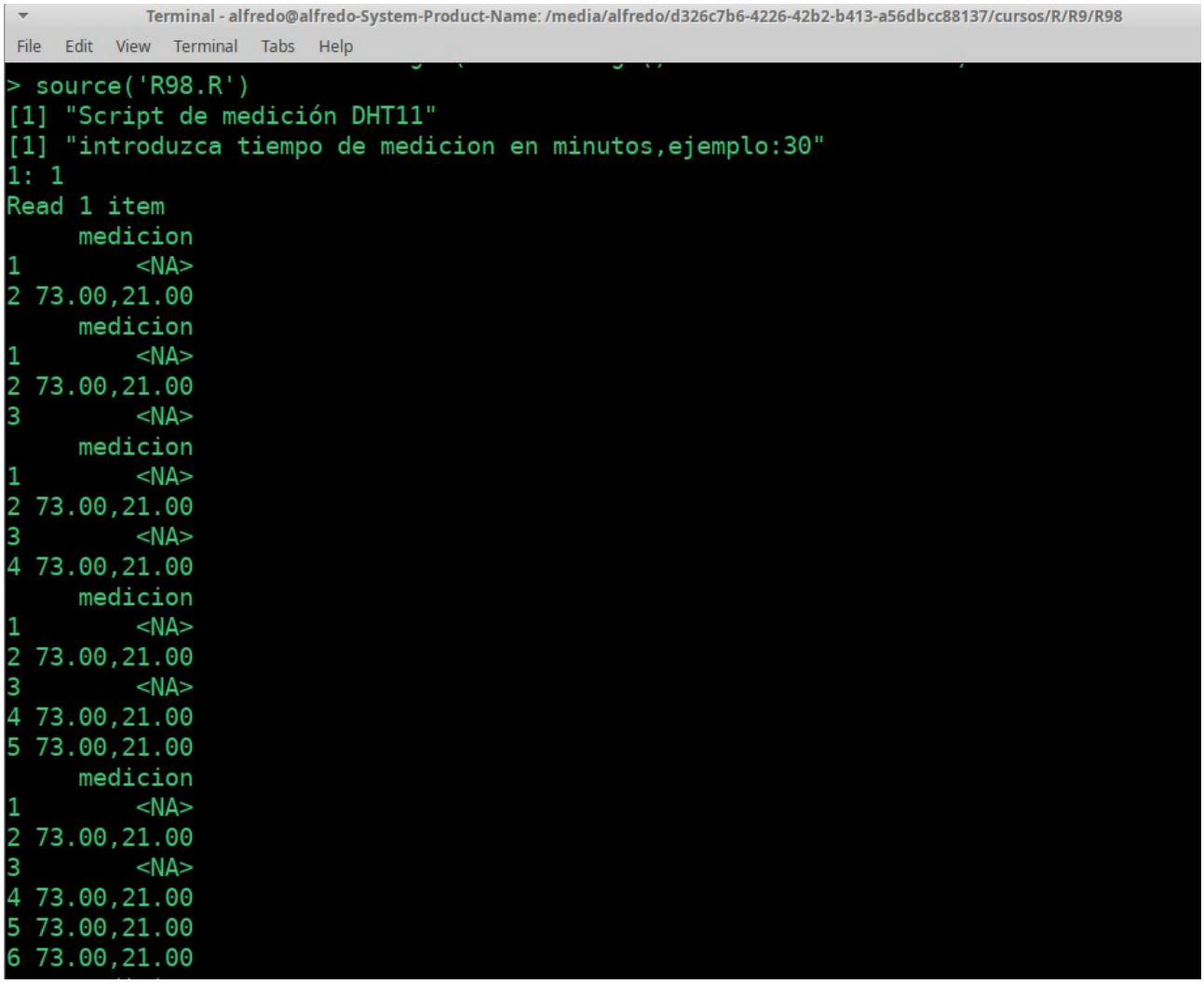


Figura 3

En la próxima clase discutiremos sobre las funciones utilizadas en el script y como podemos reprocesar los datos para un mejor manejo.

Módulo 9 – clase 9

Análisis del script para adquisición de datos

Recordemos en primer lugar el código del sketch utilizado para realizar la lectura del DHT11 desde arduino

```
// Incluimos librería
#include <DHT.h>

// Definimos el pin digital donde se conecta el sensor
#define DHTPIN 7
// Dependiendo del tipo de sensor
#define DHTTYPE DHT11
// Inicializamos el sensor DHT11
DHT dht(DHTPIN, DHTTYPE);

void setup() {
// Inicializamos comunicación serie
```

```
Serial.begin(115200);

// Comenzamos el sensor DHT
dht.begin();
}

void loop() {

// Esperamos 5 segundos entre medidas
delay(5000);

// Leemos la humedad relativa
float h = dht.readHumidity();

// Leemos la temperatura en grados centígrados (por defecto)
float t = dht.readTemperature();

// Comprobamos si ha habido algún error en la lectura
if (isnan(h) || isnan(t) ) {
Serial.println("Error obteniendo los datos del sensor DHT11");
return;
}
Serial.print(h);
Serial.print(",") ;   // coloca una coma luego del valor de h
Serial.println(t);

}
```

Por otra parte en la tabla siguiente se muestra el script utilizado para la comunicación entre el DHT11, Arduino y nuestra PC, al que llamamos R98.R

script: R98.R	explicacion
library(serial) #Facilita acceder al puerto de serie	carga biblioteca serial
print("Script de medición DHT11")	Indica la utilidad del script
system("sudo chmod 777 /dev/ttyACM1") #esta línea puede eliminarse y activar el puerto manualmente de ser necesario	En linux habilita acceso y lectura de puerto serial. Omitir en windows y probar
print("introduzca tiempo de medicion en minutos,ejemplo:30")	Pide tiempo de medición. Introducir solo numero en segundos.
tiempomedicion=as.numeric(scan(file="",what="",nmax=1))	Inicializa variable tiempomedicion con los segundos introducidos
#Toma tiempo de inicio	
tinicial<-proc.time()[3]	Inicializa tinicial con los segundos provistos por proc.time()
#En Port va el puerto que sugiere la IDE de arduino en el menu Tools > Ports	
a<-serialConnection(port="ttyACM1",mode="115200,n,8,1",translation="lf")	configuración de la conexión serial
#Si ya hay una conexión abierta por haber medido antes la cierra y abre nuevamente	
if(isOpen(a)) close(a)	Cierra la conexión en caso qe estuviera abierta
open(a)	Abre la conexión
#creamos un data.frame al que llamamos th y donde guardaremos los datos de humedad y temperatura, tendrá una sola columna llamada medicion	
th<-data.frame(medicion=NA)	Creación de data.frame para almacenar datos
read.serialConnection(a)	Lee la información recibida por el puerto serial y que se halla en buffer de memoria
Sys.sleep(5)	Espera 5 segundos
i=1	
while(tiempomedicion*60>(proc.time()[3]-tinicial)){	Bucle while que se realizará siempre que el tiempo transcurrido sea menor que el valor con que se inicializó la variable tiempomedición.
#Cada medicion es una linea con los valores de humedad y temperatura	
#20 bytes asegura que estoy agarrando al menos una medición entera	
if(nBytesInQueue(a)>30){	Condición a tener en cuenta en función de los bytes recibidos desde el puerto serial
#Leo, guardo en variable a la que llamamos buffer	
buffer<-read.serialConnection(a)	Lee el buffer de memoria con información recibida desde DHT11 y lo asigna a la variable buffer
#buscamos en la variable buffer, los saltos los saltos de linea que ingresan por el puerto como \r	

separador<-gregexpr("\r",buffer)
 	
#nos quedamos con la parte de medición entre dos saltos de linea sucesivos, es decir, nos quedamos con una medicion y la asignamos a variable ultimo
ultimo<-substring(buffer,separador[[1]][1]+1,separador[[1]][2]-1)
 	
#Guardo en data.frame th el valor de la variable ultimo
th[i+1,1]<-ultimo
print(th)
i<-i+1
}
 	
}
	Identifica con la funciòn gregexpr, la posición de los retornos de carro: \r
	
	Inicializa la variable ultimo con un valor entre dos retornos de carro
	
	guarda el valor en el data.frame th
	muestra en pantalla el data.frame th

Como vimos en la clase anterior, si ejecutamos
> source('R98.R')
habiendo previamente cargado el sketch en Arduino y estando el sistema conectado a la computadora, obtendremos en la consola de R una salida como muestra la Figura 1

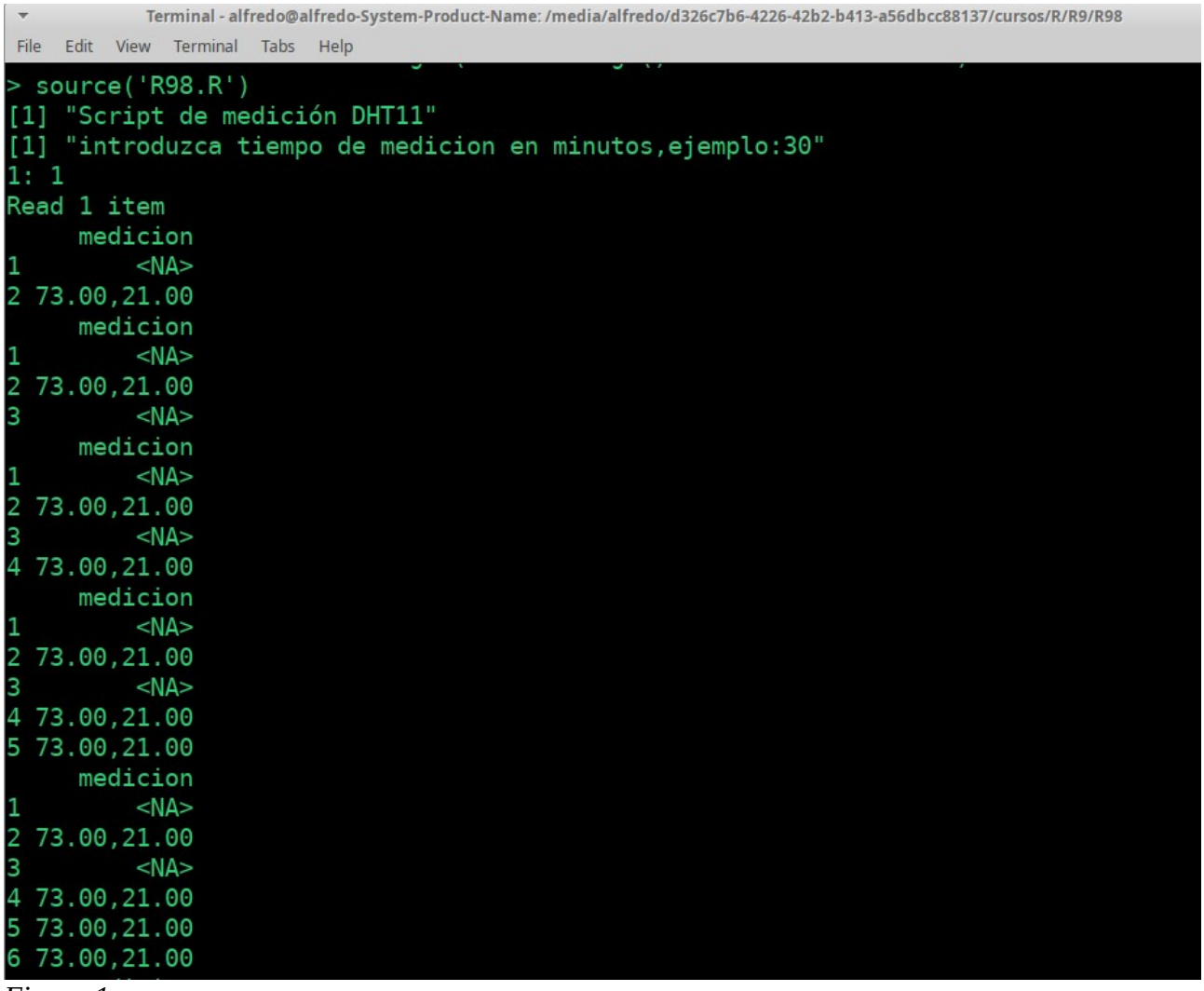


Figura 1

Como podemos ver, R no logra tomar algunas mediciones, a las que le coloca el valor NA.
Veremos por último en este módulo algunos detalles de las funciones utilizadas en el script y que nos permitirá optimizar la toma de datos, eliminando por ejemplo la falla comentada en el párrafo anterior.

Función proc.time()

Esta función nos proporciona el tiempo en segundos de diferentes procesos. Si ejecutamos

```
> proc.time()
```

```
   user  system elapsed
  0.132   0.012   6.123
```

obtenemos tres lecturas. En realidad la función da cinco valores, que los podemos pedir de la siguiente manera

```
> proc.time()[1]
```

```
user.self
     0.13
```

```
> proc.time()[2]
```

```
sys.self
    0.02
```

```
> proc.time()[3]
```

```
elapsed
201.385
```

```
> proc.time()[4]
```

```
user.child
     0.003
```

```
> proc.time()[5]
```

```
sys.child
       0
```

El valor que es de utilidad para nuestros procesos es el [3], que corresponde a los segundos desde que se inicia el espacio de trabajo.

Compruebe este funcionamiento. Para ello inicie un espacio de trabajo y simultáneamente largue un cronómetro. Deje pasar 10 minutos en el cronómetro y luego ejecute

```
> proc.time()[3]
```

obtendrá una valor cercano a 600. Seguramente habrá algunas diferencias en unidades, debido al desfasaje entre la apertura de la se sesión de R y el inicio del cronómetro. Los otros valores que nos provee la función no son de utilidad en nuestras aplicaciones.

Como ocurre en el script, si antes del bucle while, coloca el codigo

```
tinicial<-proc.time()[3]
```

estará asignando a la variable tinicial el tercer valor de la función proc.time(). Si más adelante ejecuta

```
proc.time()[3]-tinicial
```

obtendrá el tiempo transcurrido entre que inició el script y un dato tiempo.

En el script también fijamos al principio el intervalo de tiempo durante el cual se realizará la medición. Esto lo fijamos en minutos con el siguiente código

```
print("introduzca tiempo de medicion en minutos,ejemplo:30")

tiempomedicion=as.numeric(scan(file="",what="",nmax=1))
```

El script realizará mediciones hasta que se cumpla ese tiempo. Esto quedó fijado con la condición del bucle while que alberga el proceso de medición

```
while(tiempomedicion*60>(proc.time()[3]-tinicial)){
```

```
.....intrucciones de medicion....

}
```

En estas el tiempo de medición se multiplica por 60 para pasarlo a segundos y el bucle funcionará hasta que dicho valor sea menor que la diferencia: proc.time()[3]-tinicial

Función nBytesInQueue()

La función nBytesInQueue() nos proporciona el valor de los Bytes que se han ingresado por el puerto serial y que mantendrán allí, hasta que los ingresemos con la función read.serial(). Es importante conocer su valor para descargar información completa. Cuando la información ingresa por el puerto serial queda alojada en un sitio, hasta que esta es descargada e introducida en R. Se dice que la información en bytes, queda almacenada en el input buffer y luego es descargada y pasa al output buffer. Interpretelo como un recipiente que va recibiendo agua, creciendo de esta manera el volumen. Cada un dado tiempo descargamos el agua, colocando el recipiento vacio para cargar nuevamente agua. Esta descarga la haremos cada un dado tiempo. Continuando con esta analogía, la canilla que aporta el agua sería la comunicación serial, y el volumen de agua, los bytes que llegan desde arduino. Supongamos que la canilla aporta 1 litro de agua por minutos y necesitamos utilizar de a 3 litros, por ejemplo para regar cada una de nuestras plantas en el jardín. Es claro que luego de vaciar el recipiento, al colocarlo nuevamente en la canilla, debemos esperar al menos 3 minutos para poder utilizarlo. Con los bytes que llegan por la comunicación serial ocurre lo mismo. Debemos conocer qué cantidad de bytes tienen al menos una medición completa, para de esta manera cuando la descarguemos del buffer no perdamos información. Para esto nos servira la función nBytesInQueue() de la biblioteca serial.

Como hemos visto el sketch diseñado para medir temperatura y humedad, genera el valor de humedad y temperatura, separado por una coma y luego de la temperatura se coloca un retorno de carro, para que la próxima medición pase a la línea siguiente. Recordamos esta parte del código del sketch

```
Serial.print(h);
Serial.print(",") ;
Serial.println(t);
```

Esto genera en el monitor serial de Arduino, medidas como muestra la Figura 2

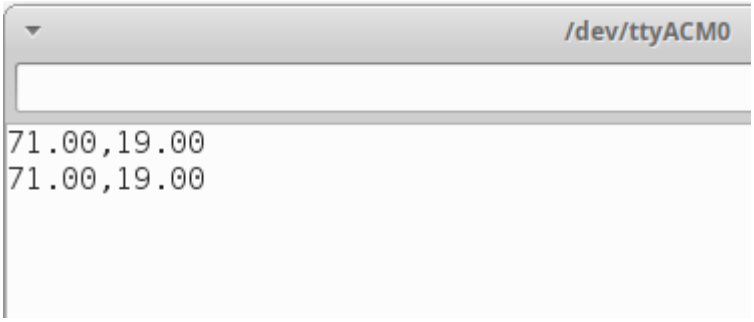


Figura 2

es decir que cada línea tiene para estos valores un formato

71.00,19.00\r

En el monitor serial no vemos \r, que es el retorno de carro, pero es información que hemos agregado con la función Serial.println() del sketch.

Como cada caracter ocupa un Byte, cada medición ocupa 13 Bytes. Si cuenta los caracteres incluyendo comas, \ y r, de la línea: 71.00,19.00\r, verá que da el valor 13. Como la información por el puerto serial se transmite Byte trás Byte, si dejamos que ingresen solo 7 Bytes en el input buffer y lo descargamos, solo tedremos parte de la medición que quedará representada por: 71.00,1.

A continuación intentaremos comprender el funcionamiento de la función nBytesInQueue().Para esto sigamos los puntos que se detallan
1- Conecte a través del cable USB el arduino con el DHT11. Asegurese que tenga cargado el sketch que hemos diseñado y se muestra al principio de esta clase. Si abre el monitor serial, debería ver que se imprime cada un dado tiempo el valor de humedad y temperatura como muestra la Figura 2. El intervalo de tiempo entre medición y medición lo estableció con la línea del sketch:
delay(5000);
que se halla dentro del sector loop(). Si logra ver en el monitor serial los datos mencionados es porque arduino y DHT11 están funcionando.
2- Cierre el monitor serial de Arduino
3- Abra R, es indistinto en que espacio de trabajo lo haga.
4- Cargue la biblioteca

```
> library(serial)
```

5- Establezca la conexión con

```
a<-serialConnection(port="ttyACM1",mode="115200,n,8,1",translation="lf")
```

Deberá controlar que el puerto que está utilizando Arduino coincida con el declarado en la función anterior
6- Abra la conexión con

```
open(a)
```

8- mida la cantidad de bytes que ingresaron con la función

```
> nBytesInQueue(a)
```

```
 n_in n_out
   13     0
```

Esto indica que ingreso una medición
9- Si repetimos la función luego que hayan pasado por ejemplo 15 mediciones obtendremos

```
> nBytesInQueue(a)
```

```
 n_in n_out
  195     0
```

Si multiplicamos el número de bytes de cada medición por el número de medicones: 13 * 15, obtenemos 195 que es el numero de bytes que vemos en la columna n_in. Por supuesto, puede verificar esto con cualquier otro número y será dificil que pueda medir exactamente 15 mediciones como lo hemos hecho aca. Aunque sabiendo que Arduino envia cada 5 segundos la información podríamos lograrlo
10- Descargamos la información del input buffer

```
> read.serialConnection(a)
```

```
[1]     "71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r"
```

11- ejecutamos nuevamente

```
> nBytesInQueue(a)
```

```
 n_in n_out
   13     0
```

Como podemos ver entre que vaciamos el input buffer y ejecutamos nBytesInQueue(a), ingreso una nueva medición.

Entonces debemos descargar la información del input buffer siempre que haya mas de 13 bytes y es conveniente que sea un número mayor de manera que estemos seguro que en la información hay una medición completa, comprendida entre dos retornos de carro. Si descargara la información solo esperando 13 bytes, podría tener una información incompleta, abarcando valores de dos mediciones consecutivas, como muestra la zona resaltada de las líneas siguientes

[1] "71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r"

Tomaremos un valor de bytes en el input buffer igual a 30, que nos asegura que tomamos con seguridad una medición, como nos muestra la salida siguiente

[1] "71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r"

Dentro del script tenemos una condición

```
if(nBytesInQueue(a)>30){
..... instrucciones.....
        }
```

donde las instrucciones incluyen la descarga de los bytes del input buffer. Vemos que no descargará nada hasta que nBytesInQueue() super el valor 30

Función gregexpr

La función geregexpr() es clave para la canalización de los datos recibidos por el puerto serial. Esta función ya fue tratada en el módulo 1 de este curso. Esta función permite reconocer un patrón dentro de un objeto de tipo string. Veamos como funciona en el script R98.R

Ya sabemos que cuando la cantidad de bytes recibidos por el puerto serial ingresen a la PC supere 30 bytes se ejecutarán en principio tres acciones que involucran los objetos buffer, separador y ultimo. Los nombres de estos objetos podrían ser otros, simplemente se utilizaron estos nombres para indicar que:

buffer: recibirá la información que se halla en el buffer del puerto serial cuando la misma se descargue con la función read.serialConnection(a). El código que le asigna el valor es

```
buffer<-read.serialConnection(a)
```

separador: es una variable que recibirá información sobre la ubicación de los separadores entre dos mediciones. Recordemos que en la consola de R, dos mediciones estarán separadas por un retorno de carro, que en caracteres está representado por \r. El codigo para inicializar la variable separador es:

```
separador<-gregexpr("\r",buffer)
```

ultimo: es la variable que guarda una sola medición, que se ubica entre dos separadores y que se asigna con el código:

```
ultimo<-substring(buffer,separador[[1]][1]+1,separador[[1]][2]-1)
```

Para comprender el funcionamiento tomemos la lectura del puerto serial obtenida recientemente, que transcribimos.

[1] "71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r"

Durante el funcinamiento de arduino con su sketch y R con su script, esta información ingresa a la variable buffer, con el código descripto recientemente

```
buffer<-read.serialConnection(a)
```

para no complicarnos con el uso simultáneo de arduino, asignemos la salida a la variable buffer

```
>  buffer<-"71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r"
```

de esta manera la variable buffer tiene 16 mediciones. Buscaremos ahora la posición de los retornos de carro, es decir es texto \r, acción que realizamos con gregexpr(), dandolo como argumentos el texto a buscar y el string, en este caso buffer.

```
> separador<-gregexpr("\r",buffer)
```

Si vemos el contenido de separador

```
> separador
```

```
[[1]]
 [1]  12  24  36  48  60  72  84  96 108 120 132 144 156 168 180 192
attr(,"match.length")
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
```

En la salida indicada con [[1]] [1], tenemos una sucesión numérica

12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192

la que nos está indicando la posición de la expresion \r dentro de la variable buffer. Si vemos el contenido de buffer nuevamente

"71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r"

vemos que las expresion \r consideradas como una sola estructura, ocupan dentro de toda la expresión las posiciones 12, 24, 36..... etc

Aclaración: cada medición recibida por el puerto serial y recibida por el script tiene el formato: 71.00,19.00\r. Es importante tener en cuenta que en tamaño de bytes la misma equivale a 13, sin embargo en caracteres representa 12. La parte \r, representa un solo caracter e indica un retorno de carro. Compruebe esto ejecutanto

```
> nchar("71.00,19.00\r")
[1] 12
```

Continuemos, entonces para quedarnos con una medición, por ejemplo la segunda utilizamos el código y asignarla a la variable ultimo utilizamos el código que involucra la función substring, también estudiada en el módulo 1

```
ultimo<-substring(buffer,separador[[1]][1]+1,separador[[1]][2]-1)
```

La función substring toma la variable buffer y selecciona el texto que se halla un caracter por encima de la posicion 1 del primer signo \t y considera hasta un caracter anterior al nuevo \t. Por esta razon como límites del texto seleccionado utilizamos

```
separador[[1]][1]+1
```

separador[[1]][1], tiene el valor 12, le sumamos 1, lo que nos permite cortar y tomar a partir del 7 como muestra la secuencia siguiente

"71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r"

y el limite superior del substring es

```
separador[[1]][2]-1
```

es decir una unidad anterior al siguiente \t, es decir toma hasta el 0 de la primer medicion como muestra la secuencia

"71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\r71.00,19.00\
r71.00,19.00\r71.00,19.00\r"

este valor es asignado a último, como podemos comprobar

```
> ultimo
```

[1] "71.00,19.00"

finalmente este valor lo asignamos a una linea siguiente del data.frame th

```
> th[i+1,1]<-ultimo
```

e incrementamos i en una unidad, de manera que la proxima medición caiga en la línea siguiente.

Una vez introducidos los datos en el data.frame th, todo estará en el espacio de trabajo y podremos manejarlo y analizarlo con las funciones de R. Ya arduino no será más parte de nuestro problema.

Si tenemos otro sensor?

En el caso que tengamos otro u otros sensores y la información que ingresa por el puerto serial es diferente al ejemplo desarrollado, el script discutido anteriormente es igualmente válido. En síntesis, este script será la base para la medida de cualquier información que ingrese por el puerto serial a la computadora.