Practica 2

Javier Aguado y Ricardo Gascuñana

## *Alumnos: Javier Aguado y Ricardo Gascuñana*

## *Fecha entrega: 03/03/2020*

# *Práctica 2 – GitHub y Ciencia Reproducible*

## 2.1. Responde a las siguientes preguntas:

#### (1) ¿Qué es la Ciencia Reproducible? ¿En qué casos se consigue un 100% de la reproducibilidad?

En las ciencias, debido a la necesidad de corroborar los logros obtenidos ante el público, es necesario establecer un **procedimiento metodológico riguroso** que explique **como se ha llegado a esos resultados**, naciendo aquí la “Ciencia Reproducible”.  
Esta es una “ciencia” que incluye todo código **reproducible mediante un ordenado**\* y que permita **regenerar el resultado final** de forma clara **con los mismos datos de partida**, sin tener nada que ver con la repetibilidad, la cual busca llegar a resultados distintos con datos de entrada distintos pero un mismo procedimiento.  
Los casos que llegan al 100% de la reproducibilidad son muy escasos, ya que **los códigos no suelen perdurar**, encontrándose únicamente el 17% de los datos disponibles tras un período de un año según diversos estudios. Esto suele ser debido a la eliminación del código por parte del usuario y a cambios constantes en los módulos necesarios para la ejecución del “Script”.  
Para llegar a ese **100%** la mejor opción se encuentra en los **repositorios on-line**, donde se puede subir código en formatos abiertos y de forma gratuita, de manera que estos cambios se pueden ir actualizando por la comunidad.

#### (2) ¿Cuáles de los beneficios que se sugieren son para ti los más importantes? Justifica tu respuesta

* **La utilización de código permite la automatización: ejecución de tareas repetitivas sin esfuerzo:** El hecho de guardar los códigos y de poder encontrarlos de forma abierta ahorra una gran cantidad de tiempo, sobre todo en aquellos trabajos donde es necesaria la creación de decenas y cientos de archivos con el mismo procedimiento. Esta automatización elimina errores que puede cometer el usuario al repetir el mismo proceso varias veces, o posibles errores matemáticos por pequeñas confusiones que el ordenador no comete.
* **La publicación del código facilita la comprensión del artículo y evita malinterpretaciones:** El hecho de publicar el código vuelve al estudio mucho más objetivo, ya que puede parecer muy complicado según las expresiones verbales utilizadas y recibir varias críticas sino está perfectamente expresado; la Ciencia Reproducible permite que todos “hablemos el mismo idioma” eliminando estas dificultades.
* **Ahorro de tiempo y esfuerzo al reutilizar código en otros proyectos:** La programación no está familiarizada con todo el mundo. La Ciencia Reproducible también sirve como biblioteca para aquellas personas que desean publicar un artículo o realizar un trabajo y tienen la necesidad de implementar algún “script” sin ser programadores. Esta “ciencia” ahorra la gran cantidad de tiempo y esfuerzo que se debería de emplear para llegar a realizar ese “script”, siendo necesarios únicamente conocimientos básicos para su interpretación.

#### (3) ¿Por qué no hace todo el mundo Ciencia Reproducible?

A pesar de ser sencillo en primera instancia, el hecho de aprender esta ciencia **conlleva tiempo** si se quiere hacer de manera correcta, como bien puede ser aprender a usar Git o Rmarkdown. Esto **no es un requisito para la calidad del trabajo** y por lo tanto no se suele tener en cuenta a la hora de publicar un artículo.

## 2.2. Lee el artículo “¿Por qué usar GitHub? Diez pasos para disfrutar de GitHub y no morir en el intento” (Galiano 2018) Realiza las siguientes tareas:

#### 1. Crea una cuenta en GitHub

#### 2. Instala Git: <https://happygitwithr.com/install-git.html>

#### 3. Preséntate: <https://happygitwithr.com/hello-git.html>

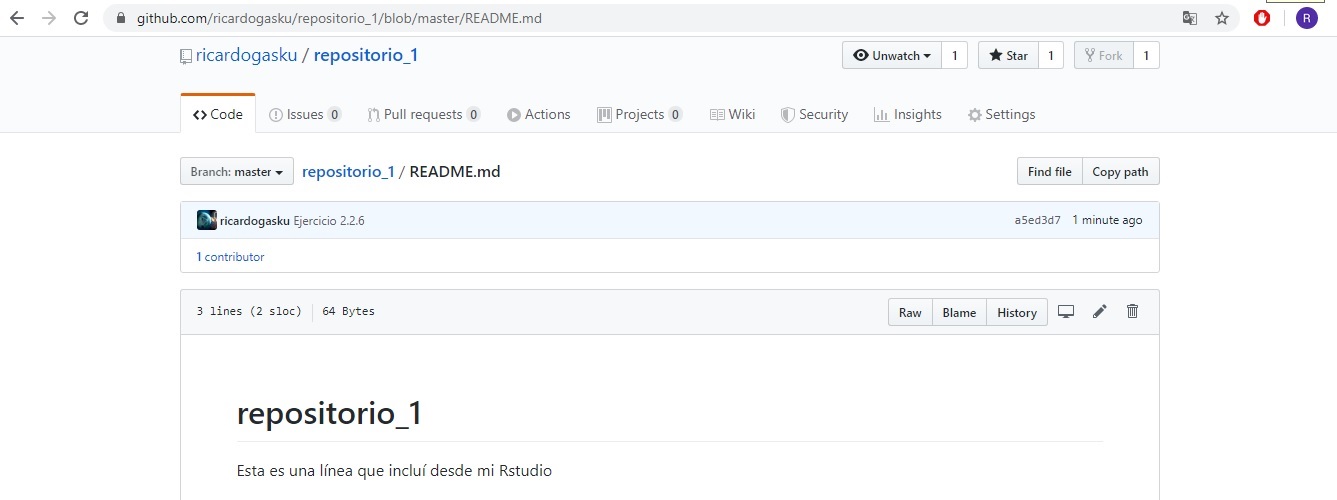
library(usethis)  
use\_git\_config(user.name = "Ricardogasku", user.email = "ricardogasku@gmail.com")

#### 4. Crea un repositorio en GitHub, haz un clon del repositorio y confirma que puedes push/pull desde la línea de comandos: <https://happygitwithr.com/push-pull-github.html#push-pull-github>

#### 5. Conecta Rstudio a Git y Github: con el repositorio de GitHub clonalo a tu ordenador usando Rstudio. <https://happygitwithr.com/rstudio-git-github.html#rstudio-git-github>.

Queda confirmado, de hecho este trabajo se inició en un ordenador, se subio con la herramienta push al repositorio y ahora se esta modificando gracias a bajarlo mediante la herramienta pull para posteriormente volver a subir las modificaciones.

#### 6. Introducir en el archivo README.md “Esta es una línea que incluí desde mi Rstudio” y comete los cambios (commit), posteriormente “push” a GitHub y comprueba que los cambios se realizaron correctamente en la nube.



readme

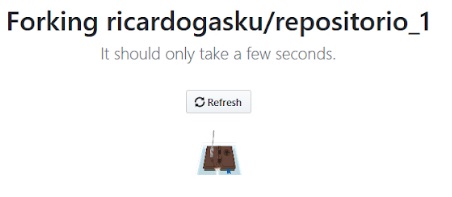
#### 7. Crea un nuevo archivo para las clases del bloque 3 y sálvalo en GitHub. ¡Ahora puedes tener todos tus códigos en la nube y trabajar fácilmente en grupo!

## 2.3. Lee “¿Por qué usar GitHub? Diez pasos para disfrutar de GitHub y no morir en el intento”.

#### Comprueba que tienes “claro” los siguientes conceptos:

* **Repositorio**  
  Es un directorio donde se almacenan los archivos del proyecto. Este se encuentra tanto en mi ordenador de forma local, como en la nube de GitHub, donde se irá actualizando cada proyecto, pudiendo además moverse a otros ordenadores.
* **Rama**  
  En Git tenemos una línea de desarrollo principal, sobre la cual trabajamos. Gracias a las ramas “branches” podemos bifurcar esta línea de desarrollo para trabajar de forma paralela y no “ensuciar” esa línea principal, desarrollando el código y pudiendo volver a juntar ambas líneas al final.
* **Pull and push**  
  Pull: Comando para acceder a un repositorio remoto y descargar todos los objetos del mismo. Seclona una copia completa del repositorio on-line en mi ordenador.  
  Push: Comando que sube a la nube los cambios hechos en mi repositorio, debe de ir acompañado de un “commit”.
* **Fork**  
  Comando que sirve para realizar una copia de un repositorio ajeno, lo encontraremos por GitHUb cuando queramos hacer una copia de algún ‘script’ que nos interese.

Desde la cuenta de Javier, se realiza un “fork” para copiar un archivo de Ricardo.



readme

## 2.4. RMarkdown. Genera un documento de html que contenga

#### 1. Todas las opciones de sintaxis básicas de Pandoc Markdown: texto plano, cursiva, negrita, superíndice, subíndice, tachado, link, ecuación en línea de texto, ecuación en bloque, diferentes encabezados.

Texto normal  
*Texto en cursiva*  
**Texto en negrita**  
Texto con Superindice  
Texto con Subindice  
~~Texto tachado~~  
[Texto con link](https://github.com/)  
Ecuación en línea:

# Encabezado 1

## Encabezado 2

### Encabezado 3

#### Encabezado 4

##### Encabezado 5

###### Encabezado 6

#### 2. Escribe fragmentos de código con las siguientes características

##### a.a. Peguen las palabas “Hola” y “mundo” en una línea nueva

"Hola"

## [1] "Hola"

"mundo"

## [1] "mundo"

##### b.b. Peguen las palabas “Hola” y “mundo” en la misma línea de código

c("Hola", "mundo")

## [1] "Hola" "mundo"

##### c.c. Peguen las palabas “Hola” y “mundo” en una línea nueva de código y que no muestren el resultado de R

"Hola"  
"mundo"

##### d.d. Peguen las palabas “Hola” y “mundo” en una línea nueva de código y que no muestren el resultado de R y que no se muestre en el documento de Word.

#### 3. Muestra una tabla y una gráfica en el archivo de salida, tomando alguno de los datos que vienen en R o alguno de sus paquetes. ¡¡Se creativo!! Usando (al menos): encabezados, explicaciones, ecuaciones, líneas de código en el texto y bloques de código.

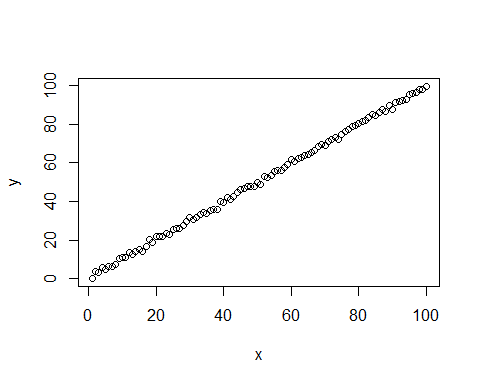
#### Gráfico

El código para la creación del siguiente gráfico, ha sido el siguiente:  
plot(pressure)



Tambien se pueden generar gráficos un poco más elavorados añadiendo algunas funciones, por ejemplo nosotros hemos añadido algunas muy sencillas:

x <- 1:100 # Asignamos a las x los valores del 1 al 100  
y <- x + rnorm(x) # Generamos datos aleatorios con probabilidad normal  
plot(x,y) # Creamos el gráfico



#### Tabla

El código para la creación de la tabla ha sido el siguiente:

library(flextable, officer)  
ftable <- flextable(iris[iris$Sepal.Length<5, 1:4]/2)  
ftable <- align(ftable, align = "center", part = "header")  
ft <- bg(ftable, bg="grey" , part="all")  
ft

En la siguiente tabla observamos las distintas dimensiones de los pétalos, las cuales han sido divididas entre 2 a través de la siguiente fórmula:   
Siendo la dimensión original del pétalo.

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width |
| --- | --- | --- | --- |
| 2.45 | 1.50 | 0.70 | 0.10 |
| 2.35 | 1.60 | 0.65 | 0.10 |
| 2.30 | 1.55 | 0.75 | 0.10 |
| 2.30 | 1.70 | 0.70 | 0.15 |
| 2.20 | 1.45 | 0.70 | 0.10 |
| 2.45 | 1.55 | 0.75 | 0.05 |
| 2.40 | 1.70 | 0.80 | 0.10 |
| 2.40 | 1.50 | 0.70 | 0.05 |
| 2.15 | 1.50 | 0.55 | 0.05 |
| 2.30 | 1.80 | 0.50 | 0.10 |
| 2.40 | 1.70 | 0.95 | 0.10 |
| 2.35 | 1.60 | 0.80 | 0.10 |
| 2.40 | 1.55 | 0.80 | 0.10 |
| 2.45 | 1.55 | 0.75 | 0.10 |
| 2.45 | 1.80 | 0.70 | 0.05 |
| 2.20 | 1.50 | 0.65 | 0.10 |
| 2.25 | 1.15 | 0.65 | 0.15 |
| 2.20 | 1.60 | 0.65 | 0.10 |
| 2.40 | 1.50 | 0.70 | 0.15 |
| 2.30 | 1.60 | 0.70 | 0.10 |
| 2.45 | 1.20 | 1.65 | 0.50 |
| 2.45 | 1.25 | 2.25 | 0.85 |