

Laboratory 1 - Data Driven Security

Arnau Sangrà Rocamora

Fall 2018

Contents

Sobre el laboratorio...	1
R	2
Hello World!	2
Usando la documentación	2
R Studio	3
Configurando las preferencias	3
Explorando el layout	5
Debug	6
Git básico	6
Creando el proyecto <i>hands on</i> con RStudio	6
Creando commits	8
Crear un repositorio remoto en Github	8
Enlazar los repositorios local y remoto	10
Push al repositorio remoto	11
Pull from remote	12
Paquetes	12
Administrar paquetes instalados con RStudio	12
Usando los paquetes instalados	13
Experimentando con el lenguaje	15
Material Complementario	16

Sobre el laboratorio...

Data Driven Security tiene como objetivo proporcionar una introducción al campo de la seguridad informática, aunque desde una perspectiva distinta.

El objetivo final de la asignatura no es únicamente descubrir las implicaciones del análisis de los datos de seguridad de la información, sino también aportar conocimiento práctico de forma que seas capaz de, al menos, explorar datos y extraer información valiosa de manera eficaz siguiendo una metodología correcta.

A través de los diferentes laboratorios, la teoría vista en clase se pondrá en práctica conjuntamente con otras herramientas y tecnologías relacionadas con el tema. Como resultado, cada laboratorio proporcionará herramientas más útiles aunque a la vez, más complejas, por lo tanto, es especialmente importante completar los ejercicios propuestos así como entender los conceptos.

R



El lenguaje utilizado en esta asignatura es ... *R*. Tal como se ve en la teoría, hay un muchas razones para utilizar este lenguaje de programación para realizar las actividades de este laboratorio.

- Perfecto para el análisis de datos:
 - Potente
 - Numerosas bibliotecas disponibles
 - Fácil generación de gráficos
- Gran comunidad de desarrolladores
- Open source

Hello World!

Como es habitual, el primer programa para codificar con un nuevo idioma es el simple y conocido programa `_Hello World!`.

En R, el programa vendría a ser algo como la siguiente linea de código

```
print("Hello World!")
```

```
## [1] "Hello World!"
```

Una vez ejecutada, sorprendentemente, muestra en pantalla la famosa frase de bienvenida.

Usando la documentación

Cuando se programa, y especialmente cuando se utiliza un nuevo lenguaje de programación, a menudo nos encontramos en la situación de no saber exactamente cómo hacer uso de las funciones que implementan la funcionalidad que queremos utilizar.

Afortunadamente, la comunidad de los desarrolladores de R, tiende a incorporar de forma habitual una excelente documentación. Así pues, todos los *packages* de funciones incluyen instrucciones especialmente completas, incluyendo descripción de lo que hacen las funciones, los parámetros que aceptan y detalles de relativos a la implementación.

Para mostrar la ayuda de una función determinada, simplemente basta con añadir un signo de interrogación antes del nombre de la función en la consola de R:

```
# show help for 'help' function  
?help
```

Además, la documentación suele también incluir ejemplos con los casos de uso más frecuentes.

- Consultar la documentación para la función `"print()"`

R Studio



Antes de ponernos manos a la obra, escribir programas en R para analizar datos es importante familiarizarnos primero con el entorno de desarrollo que vamos a utilizar.

Aunque es posible escribir programas R en cualquier editor de programación moderno (Sublime Text, MS Visual Code Studio, vim o emacs, etc), el mejor entorno donde programar R es, sin duda, **RStudio**.

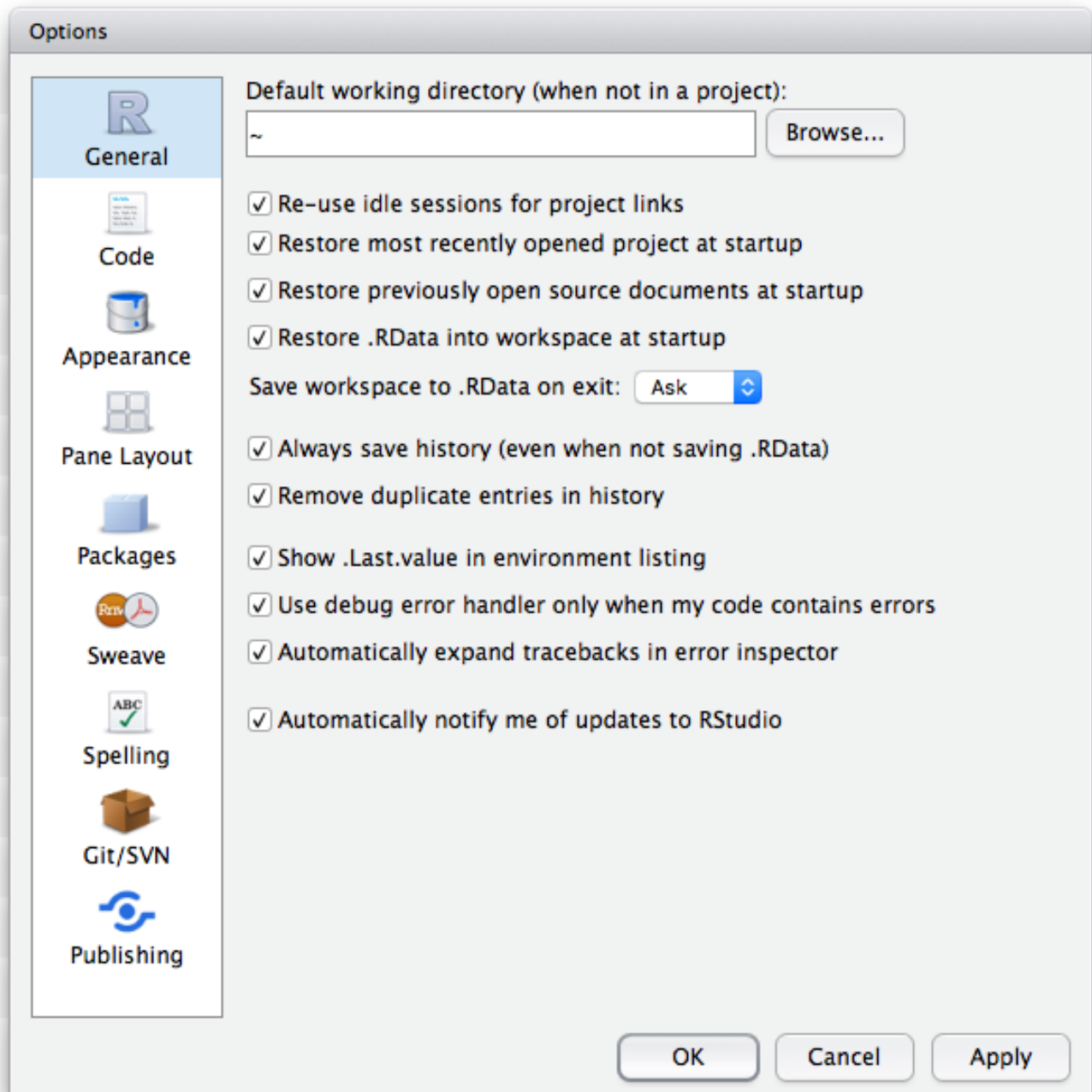
Este IDE (entorno de desarrollo integrado), disponible para todas las principales plataformas, incluyendo Windows, macOSX y muchas distribuciones de Linux, proporciona un marco único para desarrollar programas en R de una manera fácil pero potente. Entre las numerosas funciones, RStudio integra un buen editor de fuentes y un intérprete de R donde ejecutar el código.

Además, también tiene soporte para diferentes sistemas de control de versiones como git y svn, además del conjunto de herramientas más estrictamente relacionadas con el lenguaje R como el inspector de entorno, el gestor de paquetes o la trama y el visor de ayuda entre otros.

Configurando las preferencias

En primer lugar, con el objetivo de conocer el IDE (Integrated Development Environment, por sus siglas en inglés), es muy recomendable inspeccionar las preferencias de la aplicación y configurarlas en consecuencia a las necesidades personales, del proyecto o del equipo.

Como buena práctica y con el fin de prevenir futuros errores, preferiblemente, todos los equipos de los distintos miembros del equipo todos deben ponerse de acuerdo sobre una configuración mínima:



- Open preferences: (Linux/Windows: Edit -> Preferences, OSX: RStudio -> Preferences)
- En la sección *Code*, asegurarse de tener seleccionadas las siguientes opciones:
 - use *tabs as spaces*, width 2
 - set *soft wrap*
 - set *Strip trailing horizontal whitespace*
 - Saving tab: *Line ending conversion “Posix (LF)”*
 - Diagnostics tab: *Enable all R diagnostics checkboxes*

- Otras opciones útiles:
- Highlight word, currently selected line, R function calls
- show margin (80)
- RMarkdown: -Show output preview in “Viewer Pane”

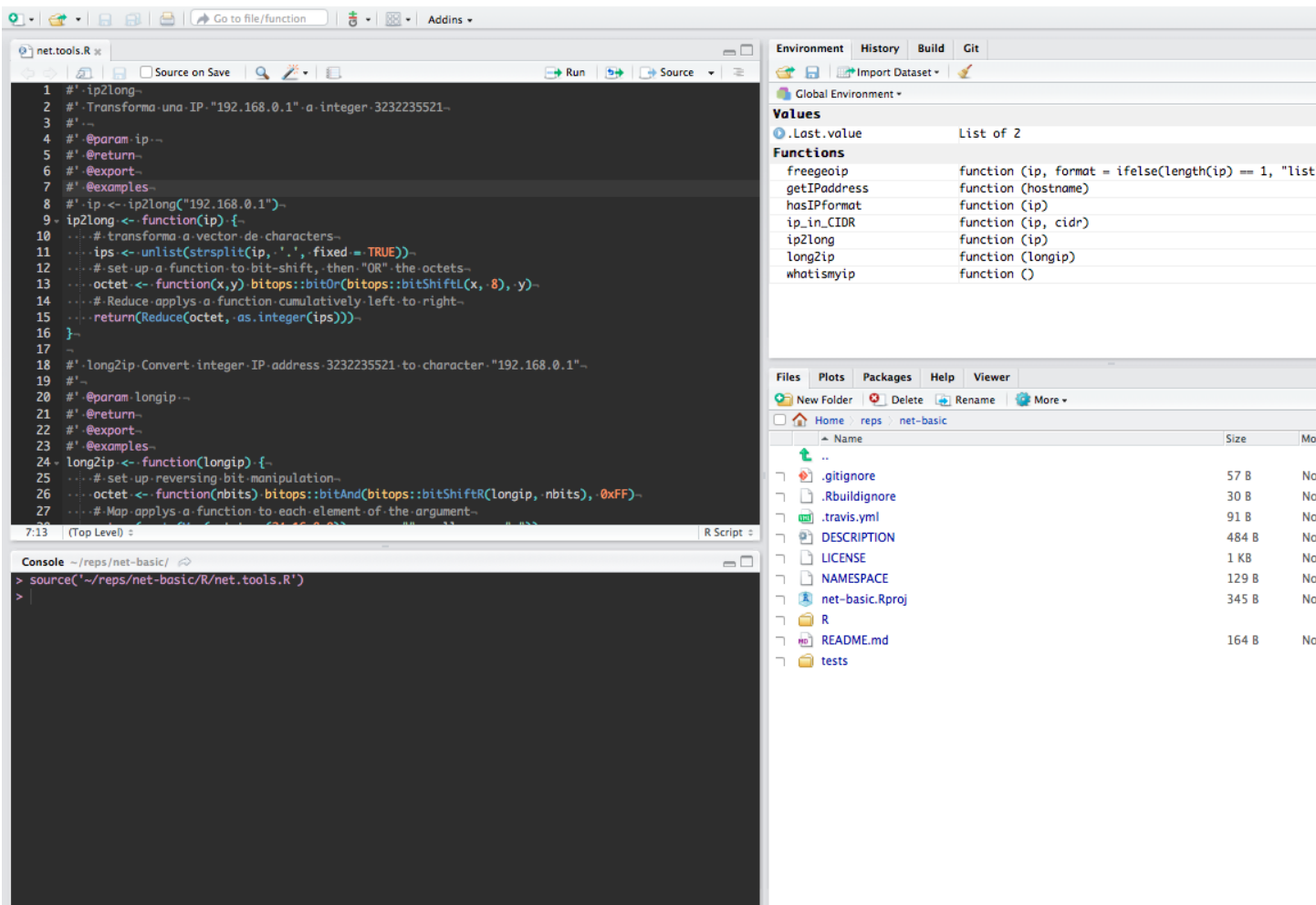
Explorando el layout

La ventana principal de RStudio esta dividida en 4 cuadrantes que contienen las principales herramientas que componen el entorno.

Es posible personalizar el diseño de la ventana para que las diferentes secciones estén dispuestas de forma diferente a la predeterminada.

- Las secciones *Source* y *Console* representan las áreas donde pasaremos la mayor parte del tiempo. Dentro del área *Source*, podemos ver todos los archivos abiertos actualmente y escribir nuestros scripts.

Por otro lado, la sección de consola nos proporciona un intérprete de R para que podamos ejecutar nuestro código tantas veces como sea necesario hasta que funcione.



No es necesario copiar y pegar las líneas de nuestro programa desde Source a Console para ejecutar. Mediante el uso de *Ctrl + Enter* / *Cmd + Enter* (macOSX) podemos evaluar la línea seleccionada desde el origen

y veremos inmediatamente la salida dentro de la consola o el resultado de la ejecución en la sección de Environment.

Por defecto, las otras dos secciones al lado de Source y Console, incluyen el inspector del Environment y el explorador de historial en un panel y el inspector de archivos, el administrador de paquetes y el visor de gráficos y ayuda en el otro.

Debug

Una de las ventajas de usar RStudio sobre otros editores es la posibilidad de fácilmente depurar nuestros programas a través del debugger integrado.

Este permite establecer puntos de interrupción, breakpoints, en cualquier punto de nuestro programa para que podamos pausar la ejecución en cualquier momento e inspeccionar el entorno en ese preciso instante de la ejecución.

- *Prueba establecer un breakpoint en cualquiera de las líneas de la funcione incluida en el fichero `r_sample.R`. A continuación, ejecuta la funcion y comprueba como se detiene la ejecución en los breakpoints insertados.*

El interprete debería detener la ejecución en la línea determinada. En este momento, la sección de entorno de RStudio debe contener todos los objetos y su valor en el mismo momento del error de ejecución. Además, también permite navegar por la pila de entornos creado durante la ejecución.

Git básico

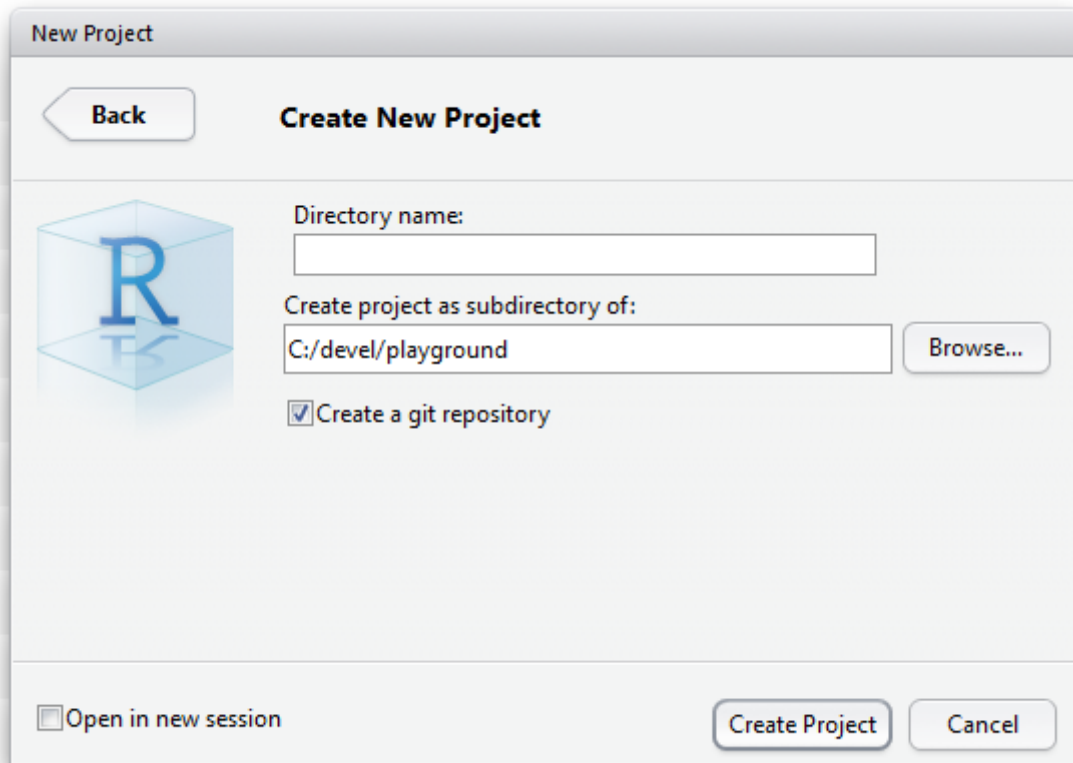


Creando el proyecto *hands on* con RStudio

Hasta ahora, hemos visto una breve introducción al lenguaje R y al RStudio IDE, el editor que usaremos para crear nuestros programas. Ahora es el momento de crear nuestro primer proyecto, es decir, el conjunto de archivos que componen nuestro código, incluyendo también los ajustes de configuración específicos y cualquier otro fichero relacionado con nuestro programa.

Aprovechando que sabemos un poco sobre Git, el VCS (Version Control System en inglés), como es probable que en algún momento estemos interesados en compartir facilmente nuestro código, poder revertirlo a una versión funcional si algo deja de funcionar, comprobar el historial de cambios, y lo que es más importante, a ser capaz de trabajar en equipo con otras personas simultáneamente, nuestro proyecto R también se basará Git.

Para ello, solo es necesario ir a File -> New Project, y seleccionar la opción “New Directory” y “Empty Project”.



- Crea un nuevo proyecto de R. Asegurate de marcar la casilla “create a git repository”.

Así de simple. Ahora tenemos nuestro primer proyecto en R, la recopilación de archivos que están relacionados con nuestro programa.

Se ha agregado un archivo con extensión `.RProj` a la raíz de nuestra carpeta, que contiene las configuraciones que establecemos para nuestro proyecto.

Dado que hemos seleccionado la casilla de verificación para crear un repositorio `git`, RStudio también ha inicializado nuestro proyecto como un repositorio local, para que podamos comenzar a realizar cambios, añadir commits.

- Crea un nuevo archivo en R (en realidad se trata de un archivo normal con extensión `.R`).
- Escribe algún programa con para añadirlo al nuevo proyecto.

Si no sabes por donde empezar, siempre puedes utilizar el siguiente código para empezar:__

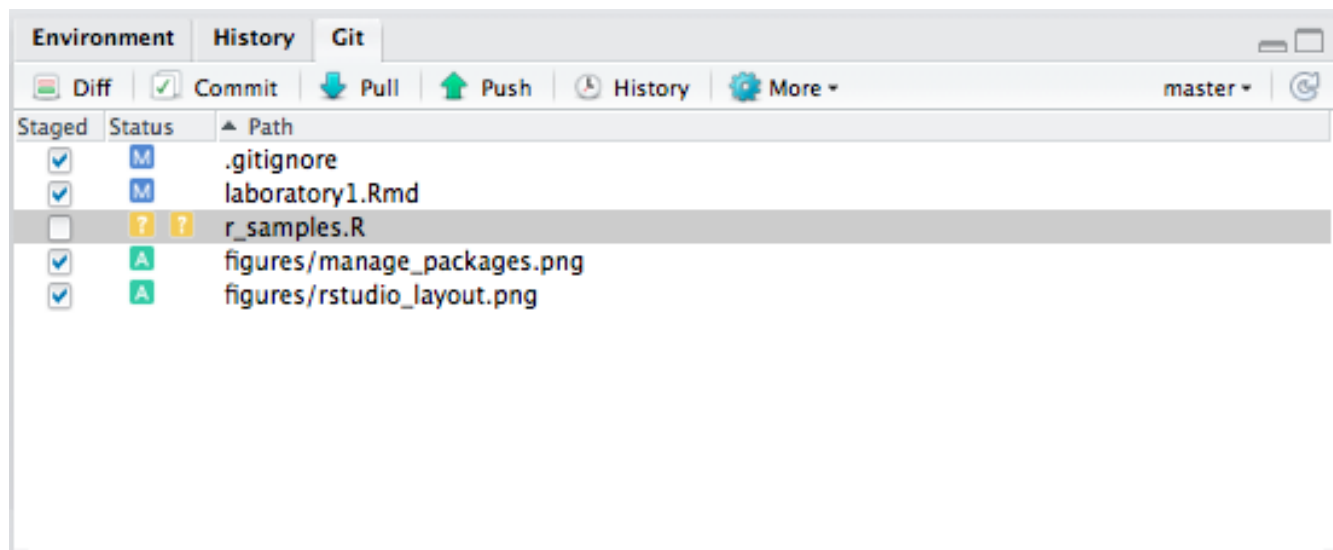
```
# This is my first R program
print("Hello World!")

subject <- "data driven security"
language <- "R"
print(paste("This is a hands on project written in",
            language,
            "for ",
            subject,
            "subject",
            sep = " "))
```

Creando commits

El *staging* área es una zona donde se almacenan temporalmente todos los cambios hasta que finalmente estos son agrupados en un único *commit*.

Una vez se ha añadido código al proyecto, para guardar los cambios al repositorio es necesario agrupar todos los cambios en un *commit*. Para ello, hay que añadir en primer lugar los archivos modificados al *staging area*.



RStudio proporciona una manera fácil de organizar los archivos dentro de la pestaña Git, donde por defecto muestra todos los archivos de su proyecto, lo que le permite añadir y quitar ficheros del *staging* área.

- Finalmente, crea al menos un nuevo commit. Recuerda incluir un mensaje relevante con respecto a los cambios realizados.

Crear un repositorio remoto en Github

Aunque git permite trabajar con un repositorio local, todas las ventajas vienen con la posibilidad de trabajar con un repositorio remoto donde enviar los commits permitiendo así que otros vean los cambios que hacemos.

Para crear un repositorio remoto usaremos *GitHub*, un servicio de repositorio freemium en Internet, donde almacenar nuestro código.

- Regístrate en Github.

Join GitHub

The best way to design, build, and ship software.



Step 1:

Set up a personal account



Step 2:

Choose your plan



Step 3:

Tailor your experience

Create your personal account

Username

This will be your username — you can enter your organization's username next.

Email Address

You will occasionally receive account related emails. We promise not to share your email with anyone.

Password

Use at least one lowercase letter, one numeral, and seven characters.

By clicking on "Create an account" below, you are agreeing to the [Terms of Service](#) and the [Privacy Policy](#).

Create an account

You'll love GitHub

Unlimited collaborators

Unlimited public repositories


✓ Great communication

✓ Frictionless development


✓ Open source community



- Crea un repositorio (público o privado). Para facilitar las cosas, con el mismo nombre que al proyecto R que acabas de crear.



[Pull requests](#) [Issues](#) [Gist](#)




Create a new repository

A repository contains all the files for your project, including the revision history.


Owner

Repository name


 arnsangra /

Great repository names are short and memorable. Need inspiration? How about **super-duper-succotash**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None**

Add a license: **None** ⓘ

Create repository

Una vez hayas creado el repositorio, todavía sera necesario establecer la relación entre nuestro proyecto R (que es también un repositorio git) y el repositorio git remoto antes de que se puedan enviar los cambios al repositorio remoto de Github.

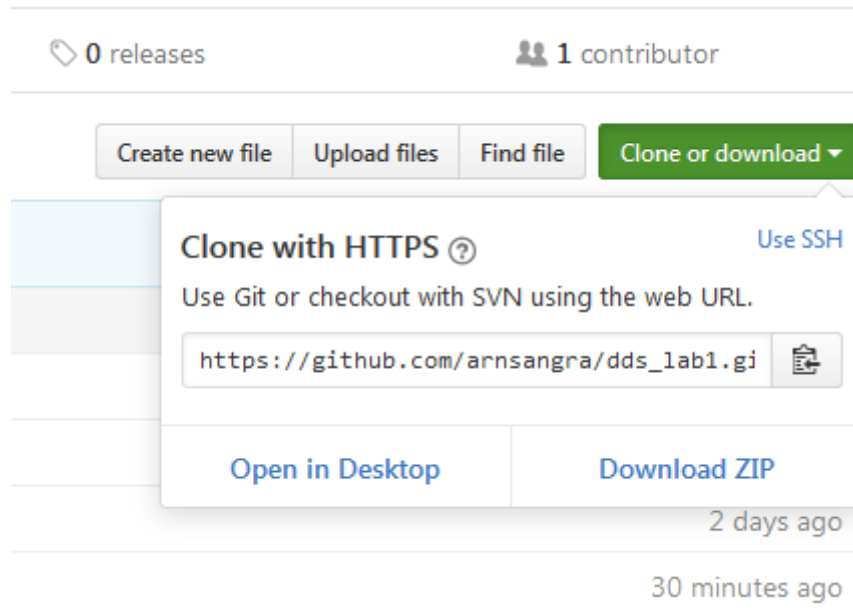
Enlazar los repositorios local y remoto

Para compartir nuestro código, por ejemplo, con cualquier desarrollador del equipo, ya que cada uno usará su propio control remoto local, es necesario vincular cada repositorio local con el control remoto central (el proporcionado por GitHub).

A pesar de la existencia de otros programas que ofrecen una interfaz gráfica que permite configurar fácilmente la dirección del repositorio remoto, RStudio todavía no permite hacerlo de forma intuitiva a menos que el proyecto se haya añadido mediante la opción de especificar una dirección de un repositorio remoto.

En este caso particular, será necesario ejecutar manualmente los comandos git en la terminal para establecer la configuración remota.

- *Obtén la dirección del repositorio remoto que acabas de crear.*



- En un terminal (shell) dentro de la carpeta del proyecto. Ejecuta la orden de git para establecer la dirección del repositorio remoto

```
# substitute '<remote repository address>' with the address copied from GitHub.  
git remote add origin <remote repository address>
```

Si la dirección remota es correcta (*https* o *ssh* preferiblemente), nuestro proyecto git tiene la dirección del repositorio remoto.

Push al repositorio remoto

A pesar de añadir la dirección del repositorio remote, aún no es posible compartir nuestros cambios.

Antes de enviar los commits locales, primero debemos especificar a qué rama queremos enviar nuestros cambios, es decir, todavía es necesario vincular el *local master branch* con el *remote master branch*.

- Ejecuta la siguiente orden de git. (ésta vincula primero las rama local actual con la remota especificada del remoto y a continuación envía los cambios a la rama del repositorio remoto).

```
git push --set-upstream-to=origin/master
```

Después de enviar el comando anterior, no solo habremos establecido la relación entre las ramas principales (locales y remotas) sino que también enviaremos los últimos cambios realizados de nuestro repositorio local al repositorio remoto.

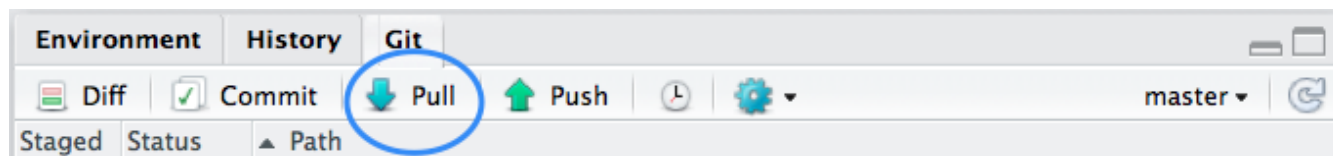


De ahora en adelante, las próximas veces que queramos enviar los commits realizados desde el repositorio local al remoto, será suficiente con hacer push. Dado que se trata de una acción muy común, la flecha verde hacia arriba desde el panel git envía los nuevos commits al control remoto para que otros desarrolladores puedan buscarlo.

Pull from remote

Por último, en caso de que estemos interesados en recuperar los últimos commits enviados al repositorio remoto en nuestro repositorio local, podemos usar la orden *pull* de git.

De forma similar al boton para hacer push, en RStudio, la flecha azul hacia abajo del panel de git hace pull desde el remoto.



Paquetes
















Los paquetes desempeñan un papel fundamental en el desarrollo de aplicaciones R. Escritos por la comunidad R, proporciona funcionalidades ya implementadas y listas para ser utilizadas dentro de nuestros programas. La mayoría de los mejores paquetes o bibliotecas están disponibles públicamente en CRAN (Comprehensive R Archive Network en inglés), un repositorio muy extenso para paquetes R.

Sin embargo, como el CRAN impone ciertos estándares y buenas prácticas para asegurar una calidad mínima, también podemos obtener paquetes de otras fuentes.

La otra alternativa principal para los paquetes R es Github. Allí podemos encontrar no sólo los paquetes que todavía no satisfacen los requisitos de CRAN, sino también las últimas versiones en desarrollo de la mayoría de los paquetes.

Administrar paquetes instalados con RStudio

Para gestionar los paquetes instalados, una vez más, RStudio proporciona una interfaz GUI útil que facilita su administración. A pesar de ello, es posible ver todos los paquetes ya instalados, instalar nuevos (desde CRAN o carpeta local), actualizar y desinstalar.

Files	Plots	Packages	Help	Viewer	
 Install	 Update	 Packrat			
Name	Description	Version			
System Library					
<input type="checkbox"/> assertthat	Easy pre and post assertions.	0.1			
<input type="checkbox"/> base64enc	Tools for base64 encoding	0.1-3			
<input type="checkbox"/> BH	Boost C++ Header Files	1.60.0-2			
<input type="checkbox"/> bitops	Bitwise Operations	1.0-6			
<input type="checkbox"/> boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-18			
<input type="checkbox"/> broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.1			
<input type="checkbox"/> caTools	Tools: moving window statistics, GIF, Base64, ROC AUC, etc.	1.17.1			
<input type="checkbox"/> class	Functions for Classification	7.3-14			
<input type="checkbox"/> cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.0.4			
<input type="checkbox"/> codetools	Code Analysis Tools for R	0.2-14			
<input type="checkbox"/> colorspace	Color Space Manipulation	1.2-6			

- *Instala algunos paquetes esenciales, que no formen parte de R:*
 - xml
 - lubridate
 - stringr

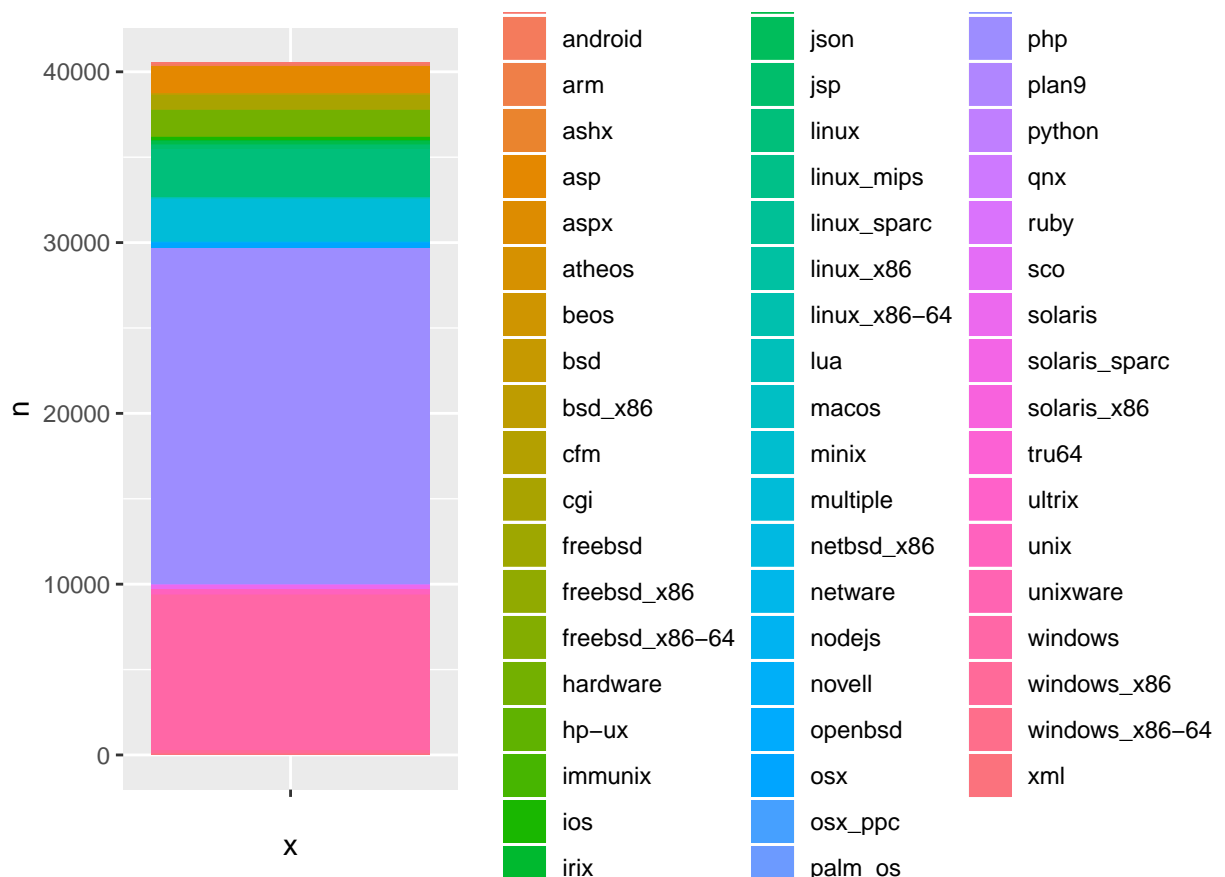
Usando los paquetes instalados

Una vez que se ha instalado un paquete, con el fin de utilizar las funciones que exportan, es decir, las características que queremos utilizar, primero debemos cargarlos en la memoria para poder usarlos.

Para ello, basta con incluir el paquete instalado en nuestro programa.

```
library("ggplot2")
# now we can use functions exported by this package

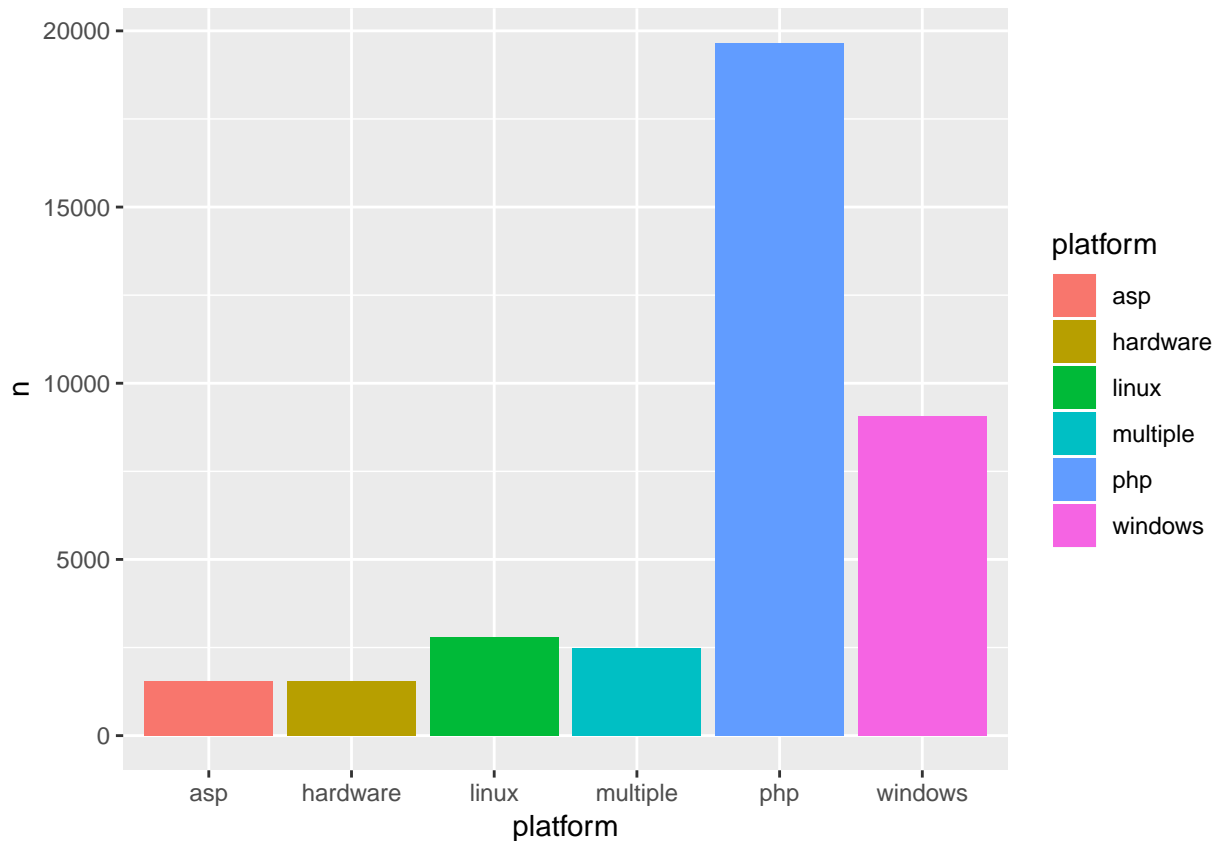
url <- "https://raw.githubusercontent.com/offensive-security/exploitdb/master/files_exploits.csv"
url2 <- "https://raw.githubusercontent.com/offensive-security/exploit-database/master/files.csv"
exploitdb <- url
download.file(exploitdb, destfile = "db")
db <- read.csv("./db", header = T)
db_aggr <- dplyr::count(db, platform, sort = T)
ggplot(db_aggr, aes(x="", y=n, fill=platform))+ geom_bar(width = 1, stat = "identity")
```



De esta manera, podremos llamar a las diferentes funciones exportadas desde nuestro código.

Por ejemplo, usando dplyr y ggplot, podemos ver fácilmente las plataformas con más vulnerabilidades

```
db_aggr <- dplyr::count(db, platform, sort = T)
ggplot(head(db_aggr), aes(x=platform, y=n, fill=platform)) + geom_bar(stat = "identity")
```



Experimentando con el lenguaje

Hasta ahora, hemos visto los principios básicos sobre los que vamos a construir los futuros laboratorios. Para familiarizarse con el mundo R, es necesaria la práctica.

Puesto que los siguientes laboratorios presuponen un cierto conocimiento de R, se debe completar la introducción de swirl, ya que representa una gran introducción al lenguaje en si de una manera interactiva y guiada.

1. *Instala el paquete Swirl (puedes utilizar tanto el panel de paquetes como las funciones R en la consola directamente):*

```
if (!require("swirl")) {  
  install.packages("swirl", repos="http://cran.rstudio.com/", quiet = T)  
}
```

2. *Carga el paquete Swirl:*

```
library("swirl")
```

3. *Ejecuta Swirl. Completa el tutorial interactivo:*

```
swirl()
```

Material Complementario

- RStudio Cheatsheet
- Git Cheatsheet
- Interactive R Tutorial
- Interactive Git Tutorial
- Git Book