INTRODUCCIÓN Y PRELIMINARES

¿QUÉ ES R?

un dialecto del lenguaje de programación S

¿QUÉ ES S?

- un lenguaje desarrollado por John Chambers (entre otros) en los Laboratorios Bell
- creado en 1976 para ser un entorno de desarrollo estadístico interno
- en principio no contenía funciones para el modelado estadístico
- en 1988 se reescribió en C en su versión 3, parecida al sistema actual
- en 1998 se publicó la versión 4, utilizada hoy en día

¿QUIÉN ESTÁ DETRÁS?

- los laboratorios Bell otorgaron una licencia exclusiva de desarrollo a StatSci (ahora Insightful)
- Insightful compró S a Lucent por 2 M$ en 2004
- Insightful fue adquirida por TIBCO en 2008 por 25 M$
- la versión comercializada en la actualidad se denomina S-PLUS

FILOSOFÍA DE S

Queríamos que los usuarios pudieran empezar con un entorno interactivo donde no fueran conscientes de estar programando. Entonces, conforme sus necesidades se aclarasen y su sofisticación aumentase, tendrían que ser capaces de deslizarse gradualmente dentro del mundo de la programación, donde los aspectos relacionados con el sistema y el lenguaje se volverían más importantes.

¿Y R?

- creado en Nueva Zelanda en 1991 por Ross Ihaka y Robert Gentleman
- en 1995 R pasa a tener licencia GNU (software libre)
- en 1997 se forma el R Core Group, que controla el proyecto
- en 2000 se publica la versión 1.0.0
- en octubre de 2016, versión 3.3.1

CARACTERÍSTICAS DE R

- sintaxis y semántica similar a la de S
- se ejecuta sobre prácticamente cualquier plataforma
- presenta un desarrollo activo con actualizaciones frecuentes
- su funcionalidad se divide en paquetes modulares (`packages`)
- potentes capacidades de representación gráfica
- útil tanto para trabajo interactivo como para desarrollo de aplicaciones
- **software libre** con una comunidad de usuarios muy activa

SOFTWARE LIBRE

- ejecutar R con cualquier propósito sin coste
- acceder y estudiar el código fuente de R para adaptarlo a las necesidades del usuario
- redistribuir copias de R y devolver mejoras a la comunidad

compartir conocimiento

¿DÓNDE ESTÁ LO MALO?

- basado en tecnología de hace 40 años
- poco soporte para gráficos 3-D o dinámicos
- su funcionalidad se basa en la demanda de los consumidores y las contribuciones de sus usuarios
- gestión de memoria, velocidad y eficiencia... **se hacen progresos**
- no vale para todo (pero esto ocurre también con Java, Python, PHP, VB...)

R VS PYTHON

DISTRIBUCIÓN

Comprehensive R Archive Network (CRAN)

- sistema **"base"** (Linux, Mac, Windows...)
- todo lo demás: **paquetes**

¿QUÉ HAY EN EL SISTEMA BASE?

- el paquete `base`, necesario para ejecutar R y que contiene funciones básicas
- otros: `utils, stats, datasets, graphics, grDevices, grid, methods, tools, parallel, compiler, splines, tcltk, stats4`
- recomendados: `boot, class, cluster, codetools, foreign, KernSmooth, lattice, mgcv, nlme, rpart, survival, MASS, spatial, nnet, Matrix`

BIBLIOTECAS O PAQUETES

- 4000 paquetes en CRAN
- muchos paquetes asociados al proyecto Bioconductor
- muchos paquetes "perdidos" en sitios web personales
- muchos paquetes en Github y Bitbucket

cómo escribir y organizar un paquete

INSTALACIÓN DE UN PAQUETE

```
install.packages("swirl")
```

USO DE UN PAQUETE

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

INSTALACIÓN DE R

instalación de R en Windows, Coursera

RSTUDIO

- Interfaz gráfica de usuario para R multiplataforma (Windows, Linux, Mac)
- Permite abrir varios scripts a la vez y ejecutar código seleccionado
- Muestra espacio de trabajo, historial, objetos del espacio de trabajo...
- Integra ayuda, gestión de librerías, etc.

posibilita desarrollar cómodamente investigación reproducible con `knitr` y `rmarkdown`

INSTALACIÓN DE RSTUDIO

instalación de RStudio en Windows, Udacity

ESTRUCTURA DE UN PROYECTO

hay muchas posibilidades, pero **lo importante es acordar una**

una ayuda: `project template`

install.packages("ProjectTemplate")

SWIRL

aprendamos R en R: cursos autoguiados con `swirl()`

- `play()`: sale del entorno de formación y deja al usuario "jugar" con la línea de comandos de R
- `nxt()`: vuelve al curso después de salir con `play()`
- `bye()`: abandona el entorno
- `swirl()`: entra en el entorno

EJERCICIO

```
> install.packages(swirl)
> library(swirl)
> install_course_github('icane','Programando_en_R')
> swirl()
```

bloques básicos de construcción

archivos y espacio de trabajo

GIT

git + github + R, Hadley Wickham

Git es un sistema de control de versiones: registra cambios al código y comparte esos cambios con otros

normalmente se empareja con GitHub, un sitio web que permite compartir código con el mundo, solicitar mejoras y registrar incidencias

Git + GitHub es el sistema de control de versiones más usado entre los desarrolladores de paquetes de R (miles de paquetes en GitHub)

recurso: git, la guía sencilla

GIT

- permite compartir código fácilmente
- GitHub hace que el código fuente sea navegable y legible
- se puede editar el mismo archivo por varios uusarios. Git combina cambios automáticamente o muestra conflictos para su solución.

- Git permite comprobar qué se ha cambiado y deshacer errores

```
install.packages("devtools")
devtools::install_github("usuario/paquete")
```

GIT

importante: acostumbrarse a usar git desde la consola (shell)

desde RStudio: Tools -> Shell

comandos más importantes de una shell:

- `pwd`: muestra el directorio de trabajo actual.
- `cd` *directorio*: cambia al directorio especificado. `cd ..` mueve un nivel hacia arriba.
- `ls`: lista archivos en el directorio actual.

más recursos sobre shell

GIT: CONFIGURACIÓN INICIAL CON R

1. Instalación: Windows, Linux, Mac
2. Configuración de usuario

```
git config --global user.name "Miguel Expósito"
git config --global user.email "exposito_m@cantabria.es"
```

3. Crear una cuenta de usuario **gratuita** en GitHub
4. Generar una clave SSH para identificación y subirla a GitHub (**si se considera necesario**)

GIT: CREAR UN REPOSITORIO LOCAL

1. en RStudio -> opciones de proyecto -> panel Git/SVN: cambiar `Version control system` a `Git`.
2. en una shell, en el directorio de trabajo, ejecutar:

```
git init
```

3. aparecerá un panel de `git` en RStudio que lista cada archivo que se ha añadido, modificado o borrado

el comando `diff` permite visualizar las diferencias entre dos archivos o dos versiones del mismo archivo

GIT: REGISTRANDO CAMBIOS

la unidad de trabajo es el **commit**, una fotografía del código en un instante determinado del tiempo.

debe ser:

- mínimo: cambios relacionados con un único problema
- completo: debería resolver el problema que dice resolver

se compone de:

- un identificador único (SHA)
- un conjunto de cambios
- un mensaje legible por humanos
- un padre (el commit anterior)
- un autor

GIT: COMMITS EN RSTUDIO

un commit presenta dos fases:

1. se indica qué archivos o cambios se incluirán en el siguiente commit (`add`)
2. se confirman los cambios con un mensaje descriptivo (`commit`)

en RStudio, las dos fases se realizan desde la ventana abierta con la opción `commit`

GIT: MENSAJES EN LOS COMMITS

1. deberían ser **concisos pero descriptivos**: de un vistazo deben dar a entender qué se está haciendo
2. deberían describir el **por qué**, no el qué: lo que ha cambiado se puede recuperar con un `diff`. Debería proporcionarse un resumen de alto nivel enfocado en los motivos del cambio.

si hacemos esto:

- será más fácil trabajar con otros (conflictos)
- los nuevos miembros del equipo comprenderán mejor la historia del código leyendo los históricos de commits
- podremos ejecutar nuestro código en cualquier punto de la historia de su desarrollo

tu más importante colaborador: **¡tu yo futuro!**

GIT: DESHACER UN ERROR

en cambios que acabamos de hacer:

- botón derecho en el archivo en el panel de git y `revert`
- también se pueden descartar fragmentos en la ventana de `diff`: `discard chunk`

si se ha hecho commit demasiado pronto:

- enviar otro commit para enmendar: marcar `ammend previous commit` antes de enviar el commit.

si está en la historia de commits:

- navegar por la historia en el panel `history` y encontrar el identificador del comit al que se desea regresar
- ejecutar en la shell: `git checkout`

y siempre hacer commit de los cambios

SINCRONIZACIÓN CON GITHUB

1. crear un nuevo repositorio en GitHub
2. añadir el remoto a nuestro repositorio local desde la shell:

```
git remote add origin https://github.com/usuario/repo.git
```

para publicar código:

```
git push -u origin master
```

para traerse código publicado:

```
git pull origin master
```

GIT: OTRAS FUNCIONALIDADES ÚTILES

`ignore`: se añaden al archivo `.gitignore`. En RStudio, hacer click con el botón derecho en el archivo en el panel Git y seleccionar `Ignore`

`blame`: muestra el último cambio realizado a cada línea de código, quién hizo el cambio y a qué commit pertenece el mismo.

muy útil cuando se está depurando un error

`merge`: fusiona código del repositorio remoto con el local. Se intenta hacer automáticamente, pero es posible tener que resolver conflictos manualmente si dos personas han editado el mismo fragmento de código.

¡ojo! si hay muchos cambios conflictivos, un merge puede llevar un ratito...

GIT: OTRAS FUNCIONALIDADES ÚTILES

ramas o branches: son realidades alternativas de la línea temporal del código. Se suelen usar para romper con el flujo principal de desarrollo.

```
git checkout -b develop
```

ejemplo: **una rama `develop` para desarrollo y una rama `master` con código que funciona**

combinar ramas:

```
git merge develop  ## combina develop con la rama actual
```

cambiar de rama:

```
git checkout rama
```

GIT: PULL REQUESTS

se trata de una herramienta para proponer y discutir cambios de manera previa a su fusión con un repositorio

se usa para contribuir con repositorios ajenos

existe documentación en GitHub

EJERCICIOS

curso de 15 minutos de GitHub

Acceso al repositorio con la presentación

Acceso a la presentación

Fco. Javier Parra Rodríguez,Jefe de Servicio de Estadísticas Económicas y Socio-demográficas