Manejo de paquetes

Julia cuenta con un gestor de paquetes ya incluido muy poderoso y simple a la vez. Este gestor se encarga de instalar, actualizar y remover paquetes *adicionales* (es decir, los que no están en la biblioteca estándar).

En el fondo, el gestor de paquetes utiliza tecnología de control de versiones (en específico utiliza git, que fue desarrollado originalmente por Linus Torvalds para el desarrollo del *kernel* de Linux), lo que hace posible tener absoluto control sobre la versión de cada paquete instalado en nuestros ambientes o proyectos. Y por encima, Julia y sus paquetes principales externos están hospedados en GitHub.

Toda la gestión de los paquetes gira entorno a dos archivos de texto plano generados automáticamente por Julia: `Project.toml` y `Manifest.toml`. El primero contiene los paquetes que hemos agregado a nuestro ambiente (por ejemplo, cuando agregamos `IJulia` al ambiente principal) y el segundo conserva un registro de las dependencias necesarias para el proyecto.

La descarga de los paquetes se hace al directorio que elegimos cuando creamos la variable de entorno `JULIA_DEPOT_PATH` (por ejemplo, `D:\.julia`). Así que aunque tengamos ambientes diferentes, Julia solo descarga UNA VEZ cada paquete (por versión necesaria).

Este es el contenido de un archivo `Project.toml` después de haber agregado `IJulia` con el gestor de paquetes.

```
[deps]
IJulia = "7073ff75-c697-5162-941a-fcdaad2a7d2a"
```

El número que vemos es un número hexadecimal del número de *commit* en el repositorio de git que contiene el código de TODAS las versiones de IJulia.

Nuevamente, la recomendación es leer la documentación del proyecto del gestor de paquetes además de leer la que viene al respecto del módulo `Pkg` en la documentación de Julia de la biblioteca estándar.

Uso desde el REPL

1. Desde el REPL, podemos acceder al gestor de paquetes tecleando `]`. Inmediatamente, el *prompt* cambiará de decir `julia>` a, por ejemplo, `(@v1.5) pkg>`, en donde `(@v1.5)` es el nombre del ambiente en el que estamos trabajando —el ambiente base en este caso— como se muestra a continuación:

```
julia> ]
(@v1.5) pkg>
```

add

Ya en el gestor de paquetes usamos el comando `add` seguido del nombre del paquete que queremos agregar al proyecto o ambiente. Por ejemplo,

```
(@v1.5) pkg> add IJulia
```

remove

Con el comando `remove` seguido del nombre del paquete, removemos dicho paquete.

status

Este comando nos regresa los paquetes agregados. Por ejemplo:

```
(@v1.5) pkg> status
Status `D:\.julia\environments\v1.5\Project.toml`
[7073ff75] IJulia v1.23.1
```

build

Este comando seguido del nombre del paquete, corre el *script* para construir el paquete de nuevo si no se ha hecho ya.

precompile

Este comando precompila todos los módulos del proyecto. Esto es recomendado después de actualizar algún paquete.

update

Este comando seguido del nombre del paquete se encarga de actualizar el paquete o módulo a la versión más reciente disponible en el repositorio.

2. De forma programática se puede usar el gestor de paquetes (por ejemplo, en un *notebook* o un *script*). Para eso es necesario importar el módulo `Pkg` (que, recordemos, está incluido en la biblioteca estándar pero no en la `Base`).

```
using Pkg
```

Después de haber hecho esto, se tiene acceso a todos los comandos del módulo usando la notación de punto:

```
using Pkg

Pkg.add("IJulia")
```