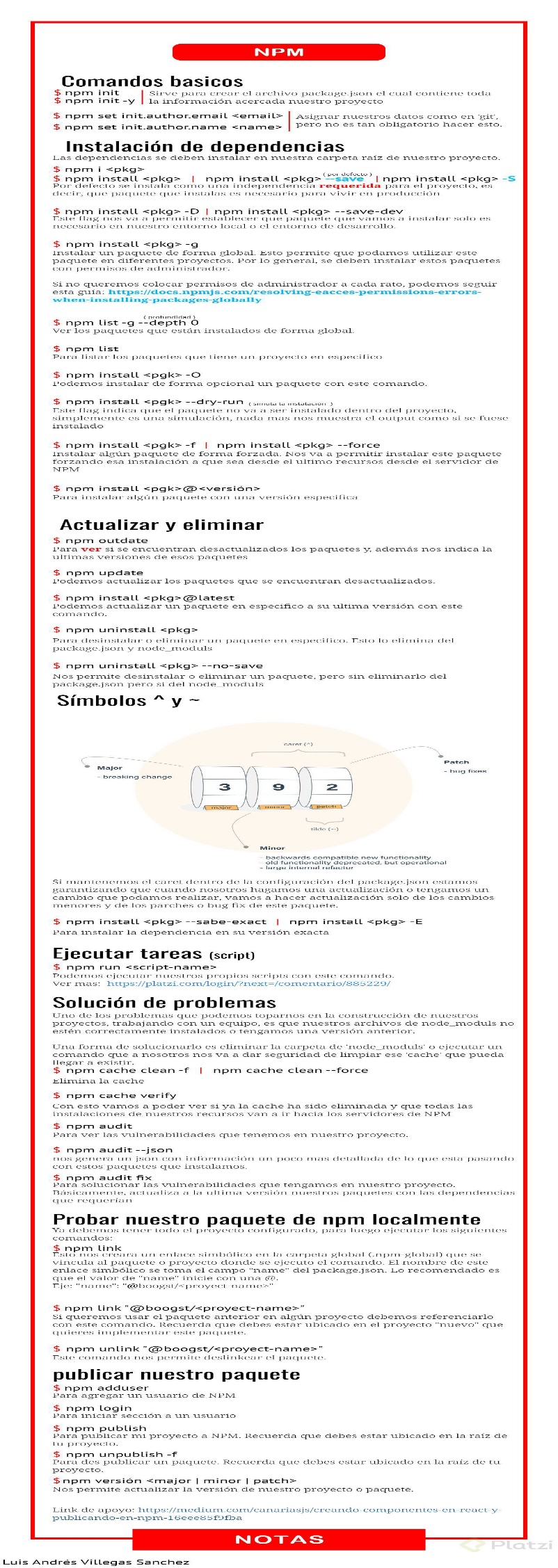
Curso de Gestión de Dependencias y Paquetes con NPM



# Acerca de NPM, paquetes y módulos

¿Qué es NPM (node package manager)? Es un gestor de paquetes, el más popular que tiene JavaScript, donde encontrarás una gran cantidad de recursos para poder implementar en tus proyectos. También vas a poder crear tus propios paquetes y compartirlos con toda la comunidad.

<https://www.npmjs.com/>

# Windows

**## Comprobar version de Node y NPM**

Una vez descargado y instalado nodejs, comprueba si se ha instalado correctamente ejecutando en una terminal (en Windows pulsa Control + R y escribe **cmd** ) el comando:

**node** **-v**

Si la salida del comando es la versión de nodejs es que se ha instalado correctamente, comprueba también que tienes npm ejecutando:

npm -v

**### Cómo actualizar NPM**

Si quieres actualizar NPM a la última versión simplemente tienes que lanzar este comando:

npm install -g npm

O si quieres actualizar NPM en MAC:

sudo npm install -g npm

También puedes actualizar NPM usando tu gestor de paquetes de la distribución de Linux si lo has instalado así.

# Mac

**Para instalar en Mac**

1. Debes de ir a [nodejs.org](https://platzi.com/clases/1763-npm/25050-mac/nodejs.org).
2. Descargar la última versión de Node.

**Para comprobar que está instalado**

1. Abre la terminal y escribe

npm -v

Esto nos da la versión que tenemos.

**Para instalar la última versión**

sudo npm install npm@latest -g```

Configuración

# Iniciar un proyecto

**Personalizando la configuración global de NPM**  
Usando los comandos de configuración de npm, podemos configurar fácilmente estos diferentes valores. Para establecer un valor globalmente ejecutamos  
npm config set <key> <value>, npm config get [<key>], npm config delete <key>, npm config list [--json], npm config edit, npm set <key> <value>, npm get [<key>]

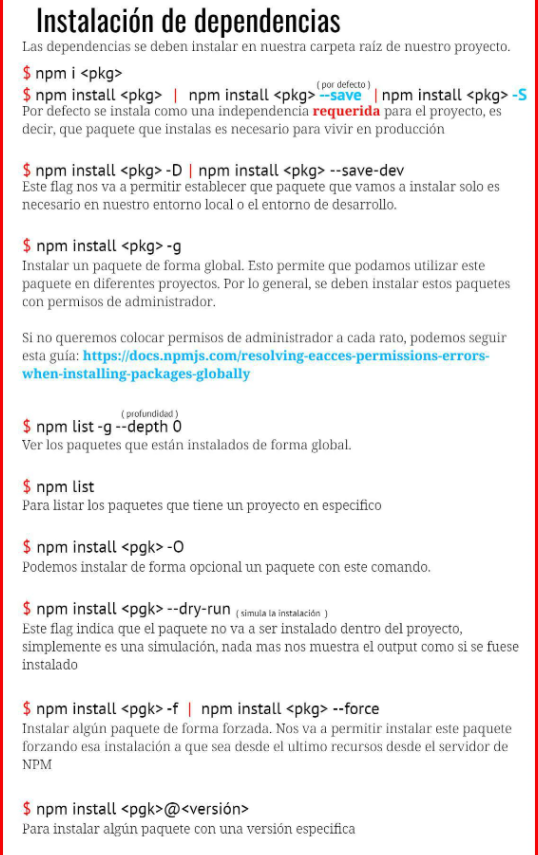
Por ejemplo, si queremos establecer el nombre, email del autor y la licencia que por defecto usaremos, puedes ejecutar  
npm set init.author.emai "[*correo@gmail.com*](mailto:correo@gmail.com)"  
npm set init.author.email "correo@gmail.com"  
npm set init.author.name "Nombre Completo"  
npm set init.license "MIT"  
Deseas verificar el valor que esta configurado utiliza el comando npm config get <key>  
Si quieres eliminar uno que creaste mal usa el comando  
npm config delete init.author.emai

Para crear el archivo de configuración con valores predeterminados usa npm init -y recomendado si ya tienes guardados datos como email, nombre y licencia



# Instalación de dependencias

No había entendido del todo la parte de las dependencias opcionales. Investigué un poco y tiene todo el sentido del mundo.  
.  
Son las dependencias que aportan algo extra al proyecto pero que no son necesarias para que el proyecto funcione. Por lo que si su instalación falla por cualquier motivo el proyecto es totalmente funcional.  
.  
Un ejemplo de esto es si quieres decorar los logs en la consola, puedes agregar un paquete como **chalk** que te ayudará a lograrlo. El proyecto va a funcionar igual asi los logs no estén decorados, por lo que este sería un paquete opcional.



Cuando queremos ver el output de una dependencia sin proceder con su instalación ejecutamos.

**npm install name\_package --dry-run**

Instalar la última versión y desde los servidores de npm:

**npm install webpack -f**

**npm install webpack --force**

**Instalar una versión exacta de un paquete:**

**npm install** [json-server@0.15.0](mailto:json-server@0.15.0)

# Instalación de dependencias con force

Este flag nos sirve para no instalar la dependencia dentro del proyecto, pero nos muestra la salida que retorna, es solo simular la instalación, de esta manera con la salida podemos decidir si instalamos o no  
**npm install (paquete) -f**  
Esta bandera forza la instalación del paquete del último recurso que tiene el servidor  
**npm install** instala todas las dependencias encontradas en el archivo package.json  
**npm fund** muestra paquetes con financiamiento  
**npm install (paquete)@0.15.0** De esta manera se instala un paquete especificando la instalación

NPM decidió agregar un nuevo comando: npm fund que brindará más visibilidad a los usuarios de npm sobre qué dependencias buscan activamente formas de financiar su trabajo.  
.  
Este comando recupera información sobre cómo financiar las dependencias de un proyecto determinado.  
.  
Si no se proporciona el nombre del paquete, enumerará todas las dependencias que buscan financiación en una estructura de árbol en la que se enumeran el tipo de financiación y la URL a visitar.  
.  
Si se proporciona un nombre de paquete, entonces intenta abrir su url de financiación utilizando el --browser parámetro de configuración.

# Actualizar y eliminar paquetes

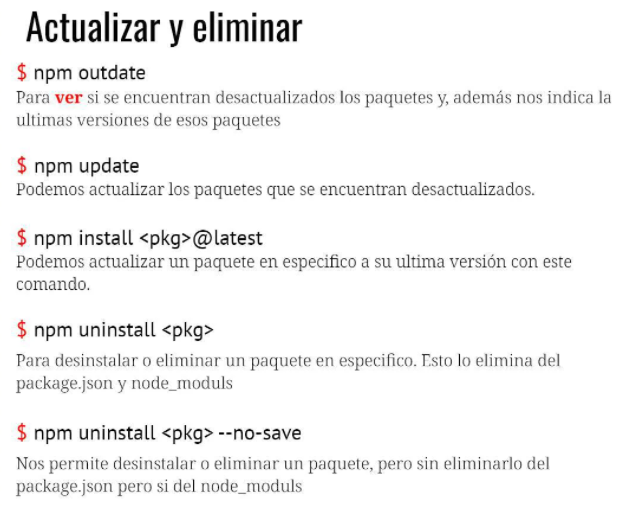
Un pequeño resumen:

Actualizar paquetes

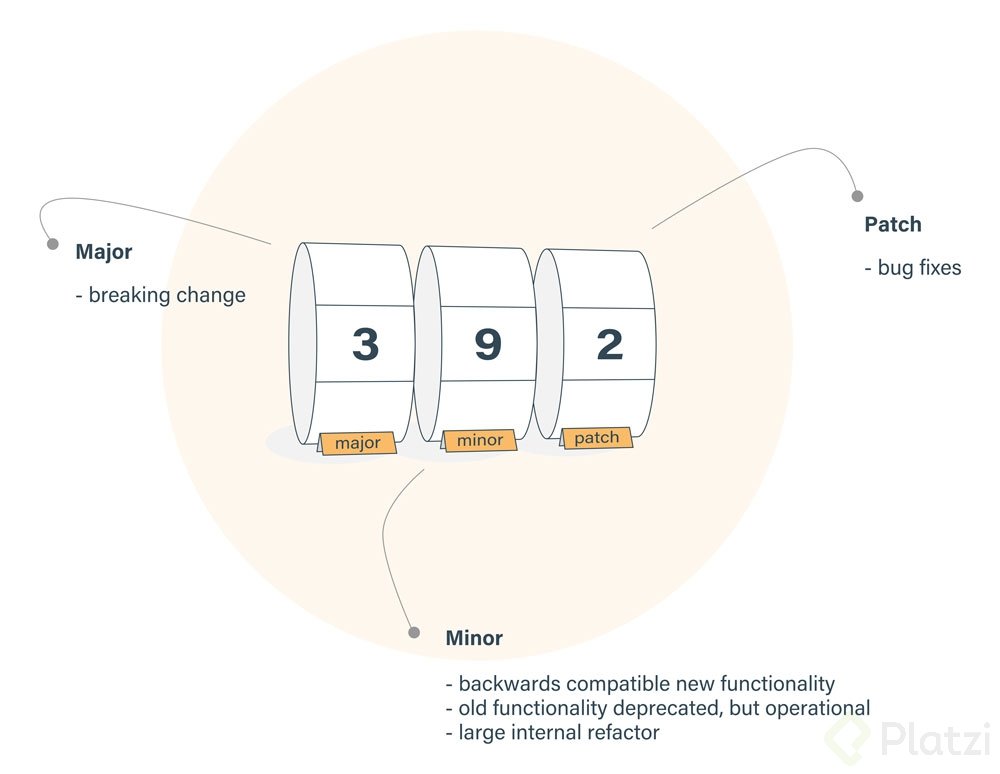
* Revisar que paquetes disponen de nuevas versiones  
  npm outdate
* Para ver un output más detallado  
  npm outdate --dd
* Actualizar los paquetes que no están en la ultima versión  
  npm update
* Actualizar un paquete especifico  
  npm install json-server@latest

Eliminar paquetes

* Eliminar un paquete de node\_modules y del archivo package.json  
  npm uninstall json-server
* Desinstalar un paquete de todo node\_modules pero no del archivo package.json  
  npm uninstall webpack --no-save



# Package lock y el uso los símbolos ^ y ~



* ^ = Si mantenemos el caret dentro de la configuración de nuestro package estamos garantizando que cuando realicemos una actualización o tengamos un cambio que podamos realizar, vamos a hacer actualización de los cambios menores y de los parches o bug fixes.  
  Para quedarnos en una sola versión eliminamos el caret.
* ~ = Establece que vamos a recibir actualizaciones o cambios solamente de los cambios que son parches o bug fixes.

Además de esos símbolos, también tenemos:

* **<** : Versión menor a la indicada.
* **<=** : Versión menor o igual a la indicada.
* **>** : Versión mayor a la indicada.
* **>=** : Versión mayor o igual a la indicada.

Los cuales se utilizan así:

"dependencies": {

"json-server": ">0.15.1",

"moment": ">=2.26.0",

"date-fns": "<2.14.0",

"react": "<=16.12.0"

}

# Ejecutar tareas

Los scripts NPM: Son comandos que podemos establecer para poder ejecutar desde la consola. Estos nos van a dar una serie de salidas según sea el caso.

Podemos crear la cantidad de scripts que necesitemos. Estos scripts van a poder correr de forma nativa dentro de nuestra terminal.

**Los scripts**  
Los scripts npm son comandos que npm ejecutará cuando se le llame con los argumentos adecuados. El poder y la sensación de esto es NO instalar los paquetes npm de manera global contaminando su entorno.

La diferencia entre las secuencias de comandos pre-reconocidas y personalizadas se basa en la palabra de run entre las etiquetas, las **secuencias de comandos custom deberán run entre npm y el nombre de la secuencia de comandos**

En función de esto, podemos diferenciar y crear diferentes tareas o scripts para ejecutar con npm.

Dado el siguiente ejemplo en el archivo package.json :

{

"name": "MyApp",

"version": "1.0.0",

"description": "",

"main": "app.js",

"scripts": {

"test": "mocha --recursive ./tests/",

"test:watch": "npm run test -- -w",

"start": "nodemon --inspect ./app.js",

"build": "rm -rf ./dist/ && gulp build"

}

}

**Podemos ver diferentes tareas a ejecutar:**

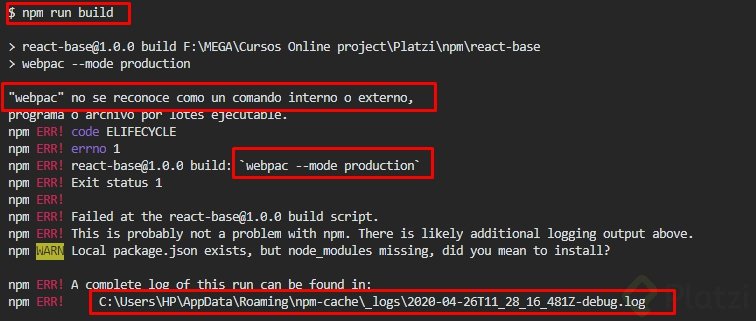
* npm test bien ya que es un script pre-reconocido
* npm run test bien ya que es una forma válida de ejecutar un script npm
* npm run test:watch también funcionaría, y se llama prueba de ejecución de npm dentro de sí mismo
* npm run build Antes de ejecutar gulp build elimine la carpeta dist que se encuentra en el directorio (suponiendo que esté en Linux o que se reconozca el comando rm )

# Solución de problemas

Cuando estés trabajando con proyectos que están usando NPM te vas a topar con una gran cantidad de posibles errores que vas a tener. Estos errores pueden ser desde la configuración, pueden ser desde el sistema operativo, espacios, no haber configurado correctamente tu GitHub, no haber establecido bien los datos del package, haber dejado un typo u algún elemento extraño dentro de esta configuración así como una serie de errores que pueden generarse, que no están ligados directamente a NPM.

* para poder activar la opción de verbose (es decir que nos muestre mayor información de lo que esta haciendo el comando)

npm run [comando] --dd

* Cuando npm nos lance algún error es recomendable ir al archivo de logs que nos muestra  
  
* En caso de que nuestros archivos de node\_module no estén bien instalados o tengamos una versión anterior lo que podemos hacer es lo siguiente:

npm cache clear --force

#Para verificar que verdaderamente se borro podemos usar

npm cache verify

* Uno de los errores que podemos tener es tener algún valor corrupto en node\_module, o tambien que la instalación no este completa de los paquetes que hemos instalado, para ello podemos eliminar el paquete con el siguiente comando:

rm -rf node\_modules #este comando eliminar la carpeta

* Otra alternativa para eliminar de forma segura una carpeta es instalando el siguiente paquete:

sudo npm install -g rimraf

* Ahora podemos ejecutar el siguiente comando para eliminar node\_module

rimraf node\_modules

#Ahora podemos volver a instalar nuestro paquetes

npm install

# Seguridad

Entendido  
Revisar si es necesario actualizar los paquetes o que contengan vulnerabilidades.

npm audit

------- resultado

found 6 low severity vulnerabilities **in** 1310 scanned packages

6 vulnerabilities require manual review. See **the** full report **for** details.

npm audit --json // Genera el resultado en json

npm update eslint-util --depth 2 // El nivel de detalle que necesita la actualización hasta las dependencias

Podemos revisar las vulnerabilidades de nuestro proyecto con:  
**npm audit**  
En caso de tener vulverabilidades, se recomienda usar el comando:  
**npm audit fix**  
Y en caso de que esto no lo solucione, podemos ir actualizandolos de uno en uno.

<https://snyk.io/>

# Crear un paquete para NPM

Ejecutar el comando para saber donde estoy ubicado

**pwd**

mkdir random-**messages**

**cd** random-**messages**/

git init

npm init

Se crea el archivo index.js en la carpeta src

// Se declara el arreglo

**const** messages = [

"David",

"Diana",

"Ana Maria",

"Isabela",

"Antonio",

"Norma"

]

//Crear función para enviar aleatoriamente los nombres del arreglo

**const** randomMsg = () => {

**const** message = messages[Math.floor(Math.random()\*messages.length)]

console.log(message)

}

// Exportar como un módulo

**module**.exports = { randomMsg }

Se debe crear una carpeta bin donde se crea ele archivo global.js (Configuración que se necesita)

#!/usr/bin/env node

// se va ejecutar dentro de bash

//Variable que llama la funcion que exportamos

let random = require('../src/index.js')

//Ejecuto la funcion

random.randomMsg()

Modifico el package.json y coloco la configuración de bin que necesito

"bin": {

"random-msg": "./bin/global.js"

},

"preferGlobal": **true**