Curso CFP

CFP Programador full-stack

NPM: Gestión de Dependencias

Agenda

- Node Package Manager
- Gestión de Dependencias
- Crear un proyecto
- Instalar dependencias
  - A partir del nombre
  - A partir del package.json
- Ubicación de las dependencias
- Eliminar/actualizar dependencias
- Listar dependencias
- Consultar dependencias desactualizadas
- Definir/ejecutar tareas del package.json
- Recomendaciones
- Manejo básico de consola

¿Qué es una librería?

- Código que hace otra persona, que usamos en nuestro código realizar una determinada tarea
- Por ejemplo si queremos hacer una suma de matrices, podemos descargar una librería que ya tenga la suma implementada
  - De esta manera nos ahorramos trabajo
- Por el momento podemos ver una librería como un conjunto de funciones que ya están implementadas por otra persona
- Se suelen usar varios nombres para definir una librería: módulos, dependencias, etc.

Node Package Manager (NPM)

- Gestor de Paquetes/Dependencias
- Se maneja por línea de comandos
- Permite descargar librerías
  - Desde un determinado lugar
- Guarda registro de las librerías descargadas
  - Junto con su versión
- Permite definir tareas
  - Facilitan el desarrollo

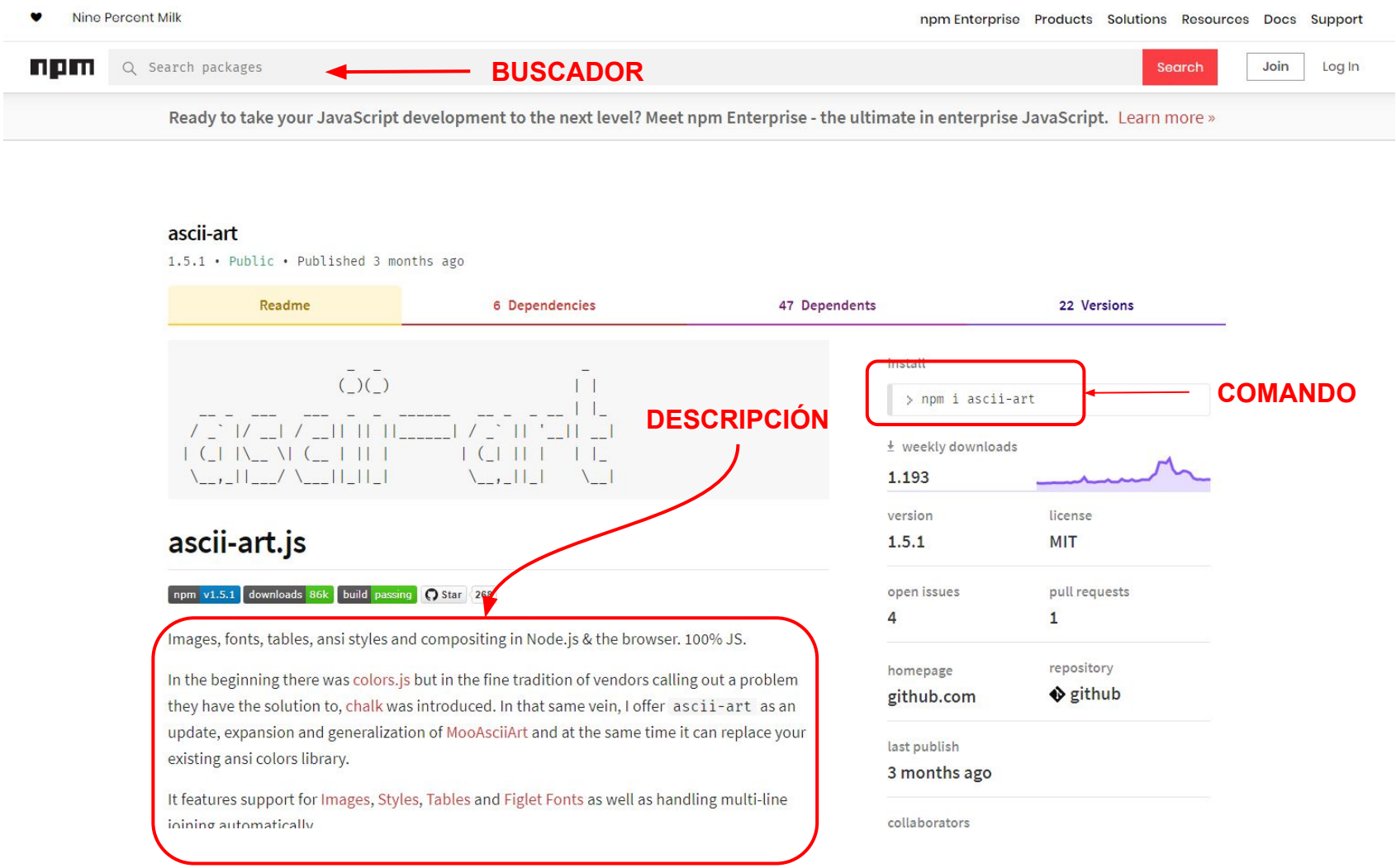
Gestión de Dependencias Introducción

- npm install readline-sync
  - “npm” es el comando del gestor
  - “install” es una de las funcionalidades de NPM
  - “readline-sync” es una librería para ingresar datos por teclado
- El comando se encarga de buscar en la nube la librería, y descargarla
- Sin este comando, al momento de hacer el “require” se obtiene el error:
  - Error: Cannot find module 'readline-sync’
- El formato del comando para instalar es
  - npm install <nombre_libreria>
- NPM tiene una página web donde poder ver las librerías

Primer paso: Crear un Proyecto

- NPM guarda en un archivo el nombre de las librerías que se van descargando → *package.json*
  - Además se guarda la versión de la librería descargada
- npm init
  - Genera el archivo package.json
- Al ejecutar el comando, la consola realiza una serie de preguntas
  - Nombre del proyecto → ej: CalculadoraMatrices
  - Descripción, autor, etc.
- Al finalizar la ejecución del comando, en la misma carpeta aparecerá el archivo con la información ingresada

Instalar Paquete



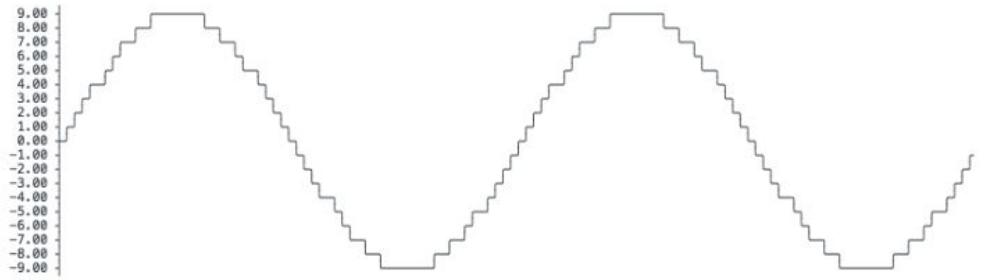
https://www.npmjs.com/package/ascii-art

Instalar Paquete - Ejemplo de Uso

asciichart

npm v1.5.9 | build passing | coverage 100% | license Unlicense

Console ASCII line charts in pure Javascript (for NodeJS and browsers) with no dependencies.
This code is absolutely free for any usage, you just do whatever the fuck you want.



Usage

NodeJS

```
npm install asciichart
```

```
var asciichart = require ('asciichart')
var s0 = new Array (120)
for (var i = 0; i < s0.length; i++)
    s0[i] = 15 * Math.sin (i * ((Math.PI * 4) / s0.length))
console.log (asciichart.plot (s0))
```

install

```
> npm i asciichart
```

weekly downloads

1.057

version	license
1.5.9	MIT

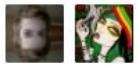
open issues	pull requests
5	4

homepage	repository
github.com	github

last publish

8 days ago

collaborators



Test with RunKit

Report a vulnerability

La misma página nos muestra cómo usar el paquete

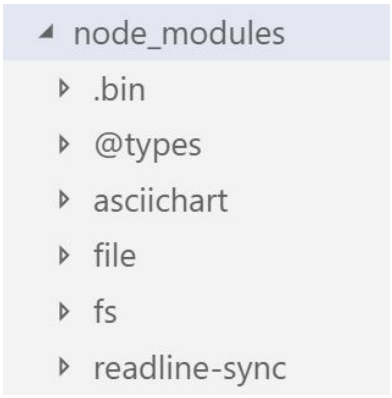
Instalar Conjunto de Dependencias

- Hasta ahora vimos como instalar paquetes → *siempre de a uno*
- Es posible que, a partir de un package.json instalar muchas dependencias con el mismo comando
- Tiene sentido cuando partimos de un proyecto ya iniciado por otra persona (trabajo en equipo)
  - Haciendo "*npm install*" se instalan todas las dependencias que figuran en donde dice "dependencies"

```
{
  "name": "ejemplo-angular-firebase",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "deploy": "ng build --prod && firebase deploy"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^7.2.6",
    "@angular/cdk": "^7.3.3",
    "@angular/common": "~7.2.0",
    "@angular/compiler": "~7.2.0",
    "@angular/core": "~7.2.0",
    "@angular/fire": "^5.1.1",
    "@angular/forms": "~7.2.0",
    "@angular/material": "^7.3.3",
    "@angular/platform-browser": "~7.2.0",
    "@angular/platform-browser-dynamic": "~7.2.0",
    "@angular/router": "~7.2.0",
    "core-js": "^2.5.4",
    "firebase": "^5.8.4",
    "ngx-toastr": "^9.1.2",
    "rxjs": "^6.4.0",
    "tslib": "^1.9.0",
    "zone.js": "~0.8.26"
  },
```

Ubicación de las Dependencias

- Al crear un proyecto, se crea el *package.json*
- Al instalar una dependencia, se guarda en una determinada carpeta → node_modules
- CUIDADO: esta carpeta no debería figurar en el repositorio de git
  - Cuando nos bajamos el proyecto, con *npm install* instalamos todo lo que figure en el *package.json*
- De esta manera el proyecto pesa menos



Dentro de esa carpeta se puede ver que cada dependencia está en su respectiva carpeta

git: Ignorando archivos

- Es posible especificar qué archivos y/o carpetas no queremos que se agreguen a nuestro repositorio
- La forma de hacerlo es especificando el nombre de los archivos, y dónde se encuentran
- El lugar en donde hay que hacerlo es un archivo especial → .gitignore
  - Este archivo se crea como cualquier otro
- De esta manera, cuando agregan los archivos con "*git add .*" los archivos/carpetas que figuren en .gitignore, no se agregan

Actualizar/Eliminar Dependencias

- Puede ocurrir que el desarrollador de la dependencia que se descargó, haya modificado la forma en que una función es invocada
  - Esto quiere decir que la versión que nos descargamos, quedó desactualizada
- Existe un comando que se encarga de actualizar las dependencias
  - npm update <nombre_dependencia>
- También puede ocurrir que descargamos una dependencia por error, o que una dependencia que antes se usaba, ahora no se usa más
  - npm uninstall <nombre_dependencia>

Listar Dependencias

- Puede ocurrir que en un proyecto uno pierda el rastro de cuales son las dependencias instaladas → comando para verlas (listarlas)
  - npm list
- Otra forma de ver las dependencias instaladas, es fijarse directo en el package.json
  - Ambas formas son válidas

```
PS C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios> npm list
ejercicios@1.0.0 C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios
`-- readline-sync@1.4.9
```

```
{
  "name": "ejercicios",
  "version": "1.0.0",
  "description": "",
  "main": "greeter.js",
  "scripts": {
    "greeter": "tsc greeter.ts && node ./greeter.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "readline-sync": "^1.4.9"
  }
}
```

Consultar Actualización de Dependencias

- Normalmente no hay que fijarse si las dependencias están desactualizadas (o no) una por una → npm outdated

```
$ npm outdated
Package      Current   Wanted   Latest  Location
glob          5.0.15   5.0.15    6.0.1  test-outdated-output
nothingness    0.0.3      git      git  test-outdated-output
npm            3.5.1    3.5.2    3.5.1  test-outdated-output
local-dev      0.0.3   linked   linked  test-outdated-output
once           1.3.2    1.3.3    1.3.3  test-outdated-output
```

COMANDO

```
{
  "glob": "^5.0.15",
  "nothingness": "github:othiym23/nothingness#master",
  "npm": "^3.5.1",
  "once": "^1.3.1"
}
```

package.json

Atajos para trabajar con NPM

- En sistemas siempre se busca hacer las cosas repetitivas en el menor tiempo posible
- Los comandos de NPM se usan bastante, por lo que existen otras formas de hacer el mismo comando
  - npm install → npm i
  - npm init -y → genera package.json con valores por defecto (sin hacer preguntas)
  - npm test → npm t
  - npm install --global imagejs → npm i -g imagejs

Definir/ejecutar Tareas en package.json

- Es muy común que una determinada tarea la realicemos de forma repetitiva
  - El mismo comando lo escribimos muchas veces
  - Ese comando puede ser largo y tedioso de escribirlo → ej: node <ruta_archivo_js>
- NPM permite definir comandos en el package.json
  - La sintaxis es “npm run <nombre_tarea>”

```
{
  "name": "ejercicios",
  "version": "1.0.0",
  "description": "",
  "main": "greeter.js",
  "scripts": {
    "prueba": "node prueba.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "readline-sync": "^1.4.9"
  }
}
```

```
PS C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios> node prueba.js
Hello, world
PS C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios> npm run prueba

> ejercicios@1.0.0 prueba C:\Users\Francisco\Documents\CFP\3. POO\Ejercicios
> node prueba.js

Hello, world
```

Demostración en Clase

- Crear un proyecto
- Implementar una determinada funcionalidad
- Correr el archivo con node
- Definir una tarea en package.json
- Correr el archivo con NPM

Recomendaciones

- No instalar dependencias que no se vayan a usar
- Intentar mantener todo actualizado
  - Para así evitar problemas de compatibilidad
- No modificar manualmente la sección de dependencias del package.json
  - Los comandos de NPM son los que modifican automáticamente esas secciones
- Acostumbrarse a usar las tareas para trabajar de forma más prolija
- Acostumbrarse a usar la consola

Adicional - Manejo de Consola

Comandos básicos porque van a empezar a trabajar más con la consola

- dir → listar los archivos y carpetas a partir de la ubicación actual
- cd → acceder a una carpeta, cambiar directorio
- cls → limpiar el contenido de la pantalla
- ren → renombrar un archivo
- copy → copiar un archivo
- del → borrar un archivo

CFP
Programador full-stack

Ejercicios

Ejercicios - En Clase

- Crear un proyecto
- Crear dos archivos con funcionalidades diferentes
- Definir las tareas asociadas y ejecutarlas

Ejercicios - Fuera de Clase

- Crear Proyecto NPM
- Buscar en la página alguna dependencia e instalarla (además de readline-sync)
- Desarrollar alguna funcionalidad que utilice la dependencia instalada
- Crear dos archivos con funcionalidades diferentes
  - Cada uno utilizando una librería diferente, sin contar readline-sync
- Definir las tareas asociadas y ejecutarlas
- Subir proyecto a GitHub y pasar link por Slack