npm

npm
Partie 1

NPM : GESTIONNAIRE DE PACKAGE

- npm est le gestionnaire de package pour Node.js
  - Node Package Manager
  - Fonctionne en ligne de commande
  - Équivalent de Composer pour PHP

- npmjs.com : répertoire des packages disponibles en Open Source
  - Possibilité de package privés en version payante

- Permet d'initier un projet Node.js

INITIER UN PROJET NODE.JS

- En ligne de commande
  - Créer un dossier `npm-demo`
  - Se placer à l'intérieur du dossier
  - Taper :

```
1   npm init
```

  - Répondre aux questions (souvent en tapant Entrée)
    - Nom de l'application
    - Version
    - Description
    - Fichier principal
    - Commande de test
    - Dépot GIT
    - Mots-clés
    - Auteur
    - Type de licence

INITIER UN PROJET NODE.JS

- Une fois toutes les questions répondues, npm présente le contenu du fichier `package.json` qu'il est sur le point de créer. Valider la création.

Exécution de npm init :

```
package name: (npm-demo)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /Volumes/Data/3iL/www/b3_nodejs/npm-demo/package.json:

{
  "name": "npm-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
```

PACKAGE.JSON

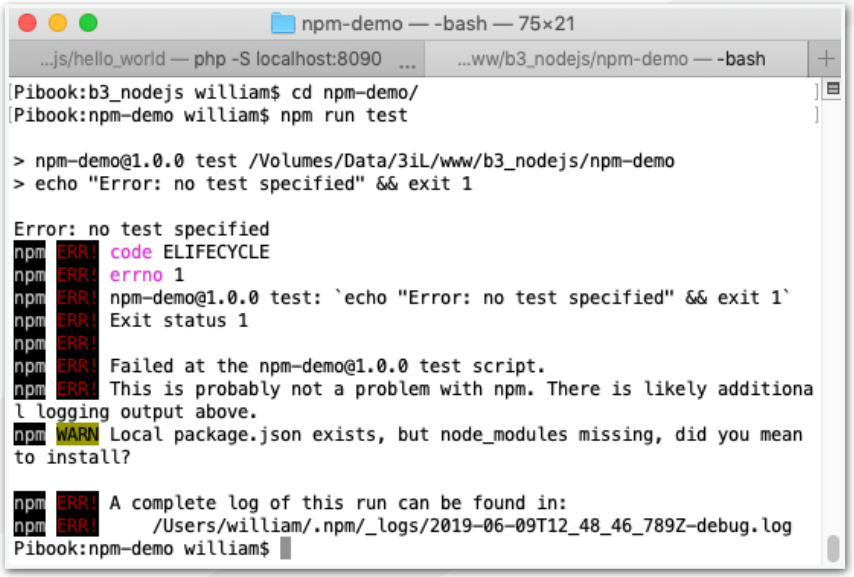
- Contenu du fichier `package.json`

package.json :

```
1  {
2    "name": "npm-demo",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC"
11 }
```

Commandes exécutables via npm run

- Zone scripts permet de définir des commandes à exécuter
- Taper `npm run test`



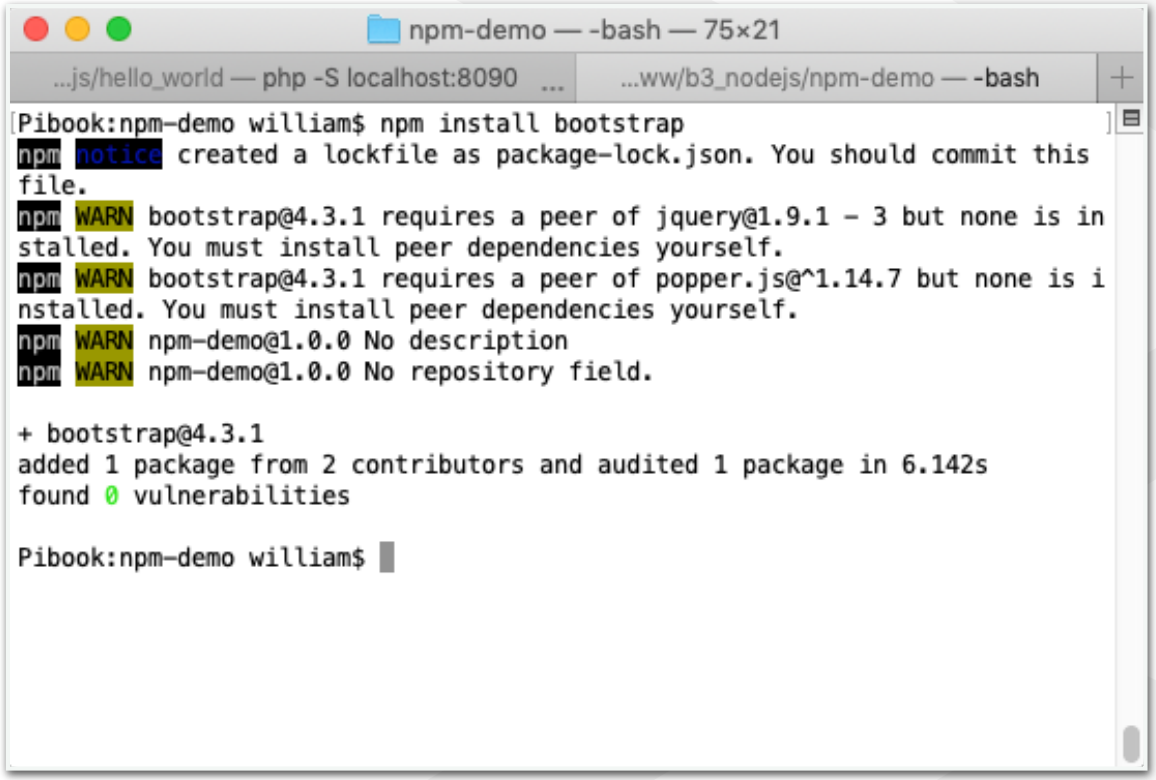
INSTALLER UN PACKAGE

- Pour installer un package, il suffit de faire :

```
npm install monPackage
```

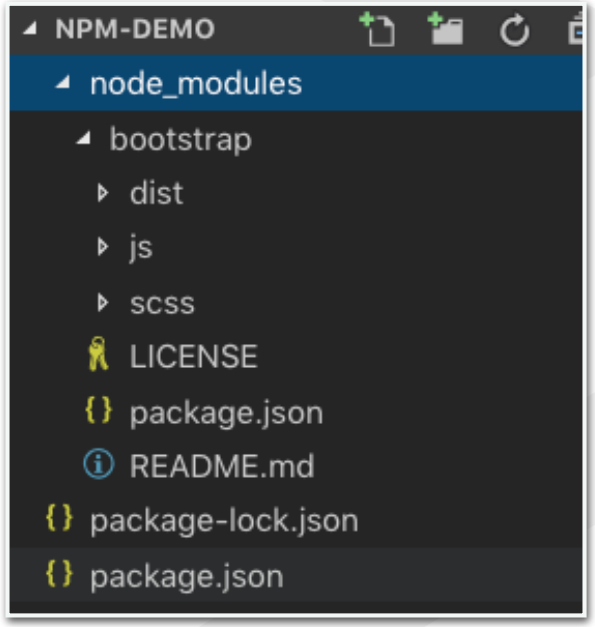
- Exemple installer Bootstrap :

```
npm install bootstrap
```



INSTALLER UN PACKAGE

- Cela a eu pour effet :
  - La modification du fichier `package.json` pour intégrer la dépendance
  - La création du fichier `package-lock.json` pour garder trace de la version exacte installée.
  - La création du dossier `node_modules` pour stocker de façon organisée les dépendances du projet.



INSTALLER UN PACKAGE

- Contenu du fichier `package.json` :

package.json :

```
1  {
2    "name": "npm-demo",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "bootstrap": "^4.3.1"
13   }
14 }
```

liste des dépendances du projet

- Existence d'une option `--save` que l'on peut encore trouver dans d'ancienne documentation. Avant npm 5.0 :
  - `--save` : enregistre la dépendance dans `package.json` (inutile depuis)
  - Sans : installe la dépendance sans la faire figurer dans `package.json`

npm
Partie 2

LES NUMÉROS DE VERSION

- Codage sur trois nombres des numéros de versions

- Numéro majeur : "grand saut de version"
- Numéro mineur : ajout de fonctionnalité sans remettre en cause le package
- Numéro de patch : correction d'erreur

LES NUMÉROS DE VERSION

- ^ et ~ sont les deux principaux symboles rencontrés dans les numéros de versions avec npm.
- Exemple avec Bootstrap (4.3.1)
  - `^4.3.1` : installera n'importe quelle version 4.x.x, ne prendra pas en compte la version 5.0.0
  - `~4.3.1` : installera n'importe quelle version 4.3.x, ne prendra pas en compte la 4.4.0
- Sans aucun symbole : `4.3.1` npm n'installera que cette version précisément.

LES NUMÉROS DE VERSION

- Utiliser @ pour installer une version précise d'une dépendance :

```
npm install jquery@2.1.2
```

- Attention :
  - npm installera bien la version `2.1.2`
  - npm inscrira `^2.1.2` dans le fichier `package.json`

- Pour installer la dernière des versions 2 de jquery :

```
npm install jquery@2
```

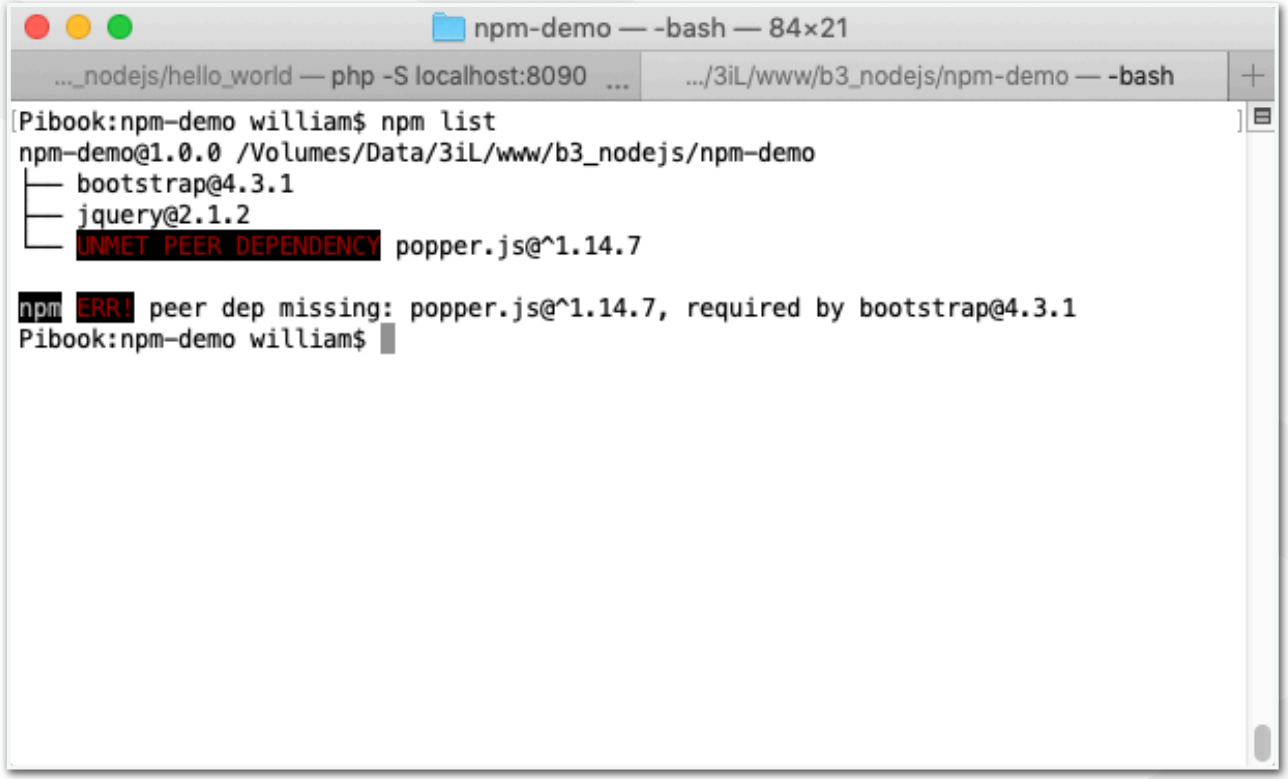
- Pour installer la dernière des versions 2.1 de jquery :

```
npm install jquery@2.1
```

- Note : actuellement jquery est en version 3.4.1

LISTER LES DÉPENDANCE

- Pour lister les dépendances : `npm list`



- Note : ici `popper.js` est déclaré manquant pour Bootstrap (non obligatoire)

RÉINSTALLER TOUTES LES DÉPENDANCES

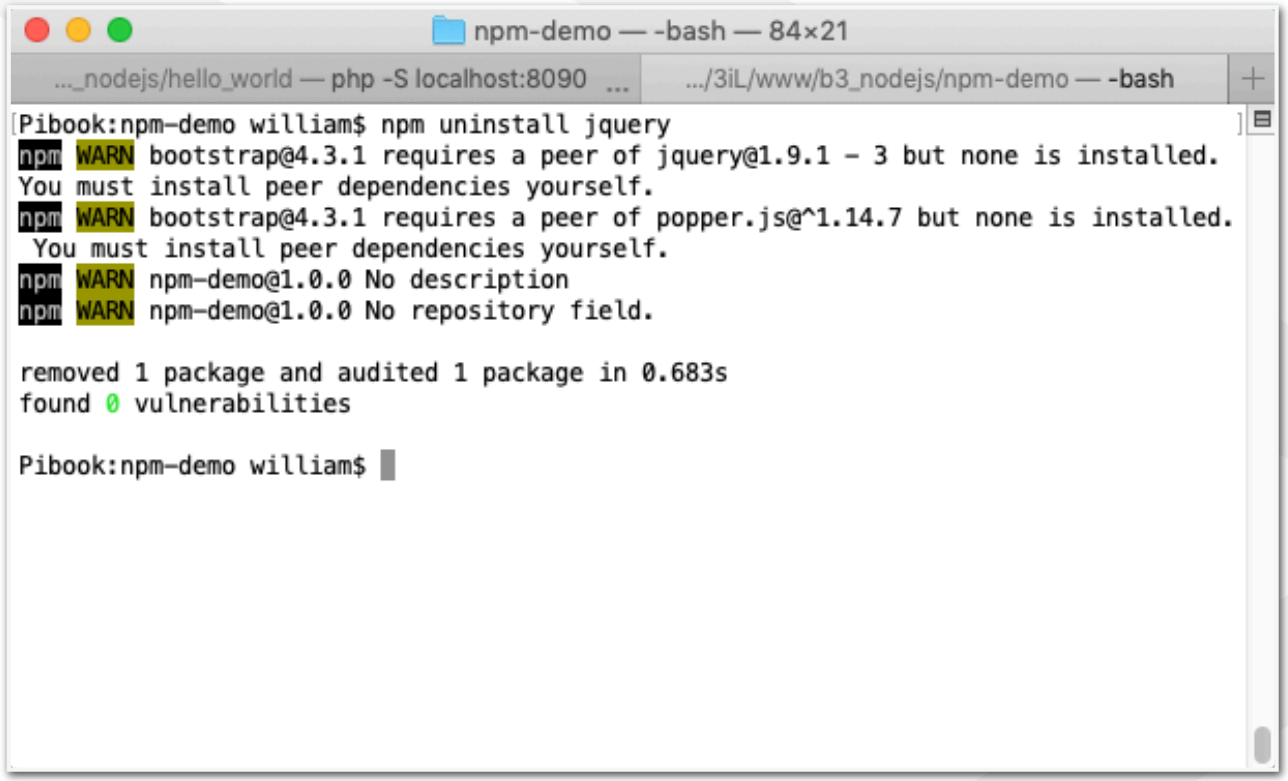
- `npm install` permet de réinstaller toutes les dépendances
  - Supprimer le dossier `node_modules`
  - Taper `npm install` et toutes les dépendances seront réinstallées

- Fonctionnement avec git
  - Dépendances inutiles à versioner dans un projet
  - Mettre le dossier `node_modules` dans `.gitignore`
  - À la récupération d'un projet un simple `npm install` permet de retrouver le projet complet
  - Évidemment : le fichier `package.json` est ultra important.

SUPPRIMER UNE DÉPENDANCE

- Pour supprimer une dépendance il suffit d'exécuter `npm uninstall` :

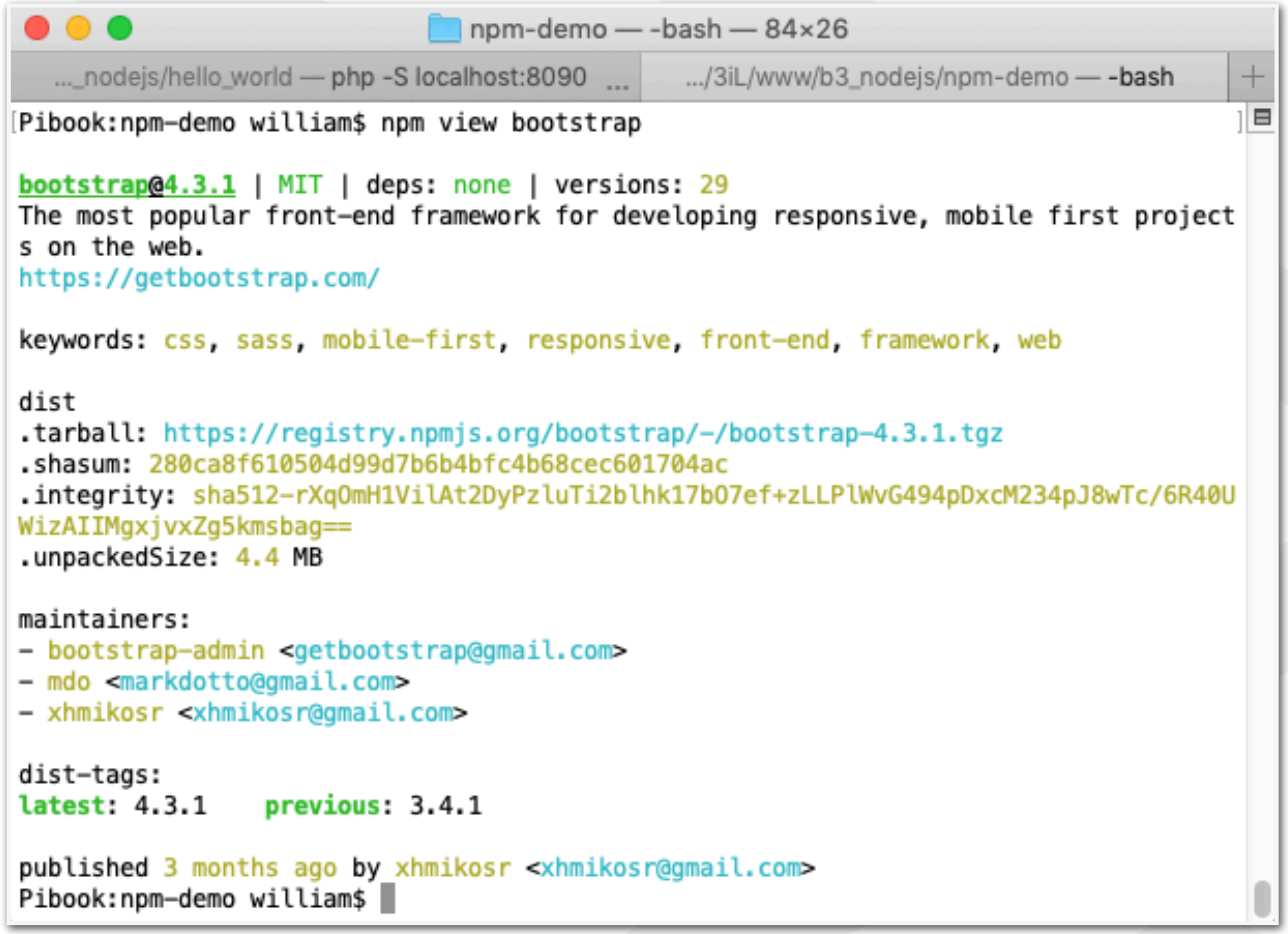
```
npm uninstall jquery
```



INFORMATIONS SUR UN PACKAGE

- Pour avoir des informations sur un package `npm view` :

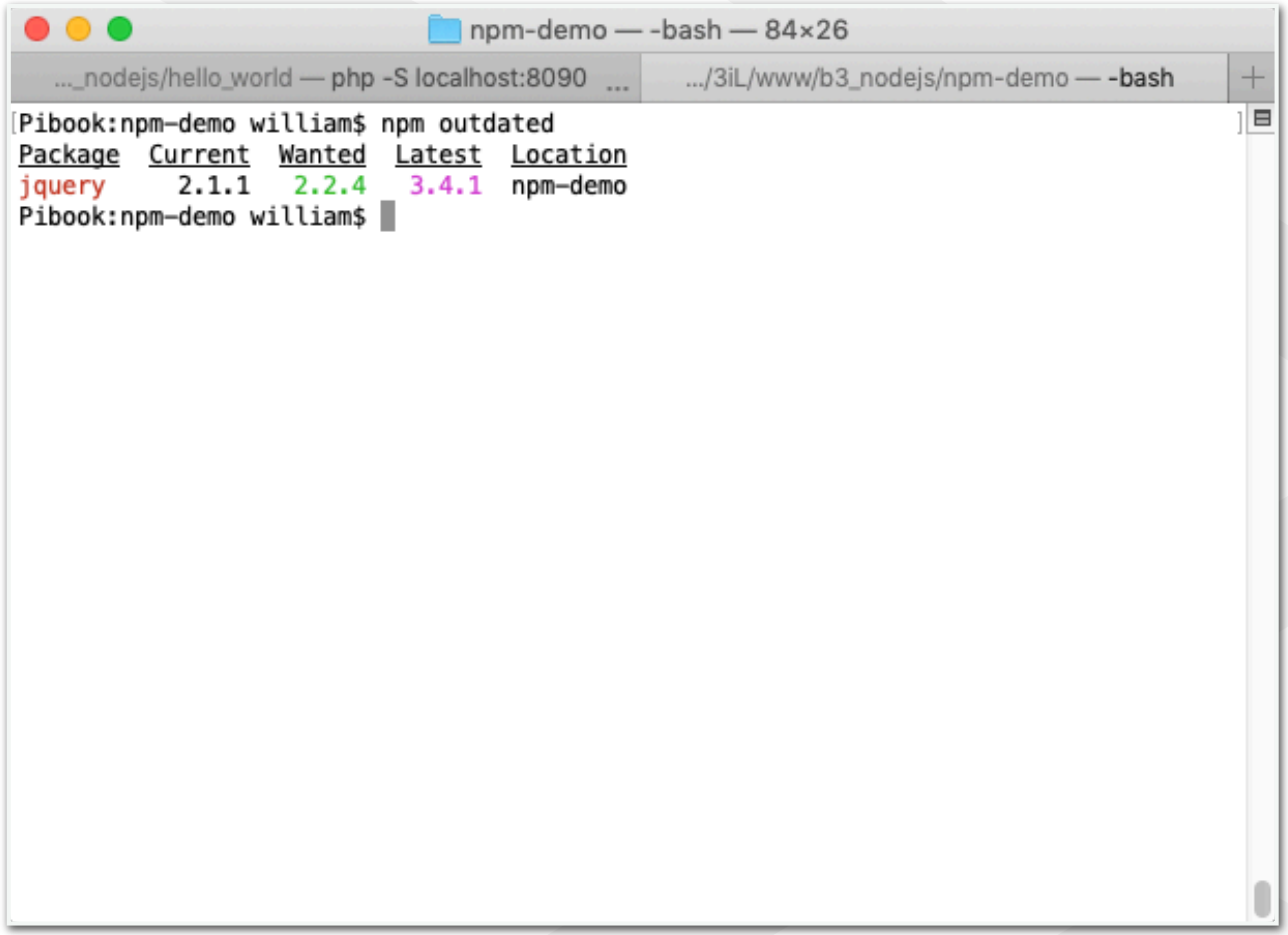
```
npm view bootstrap
```



METTRE À JOUR LES DÉPENDANCES

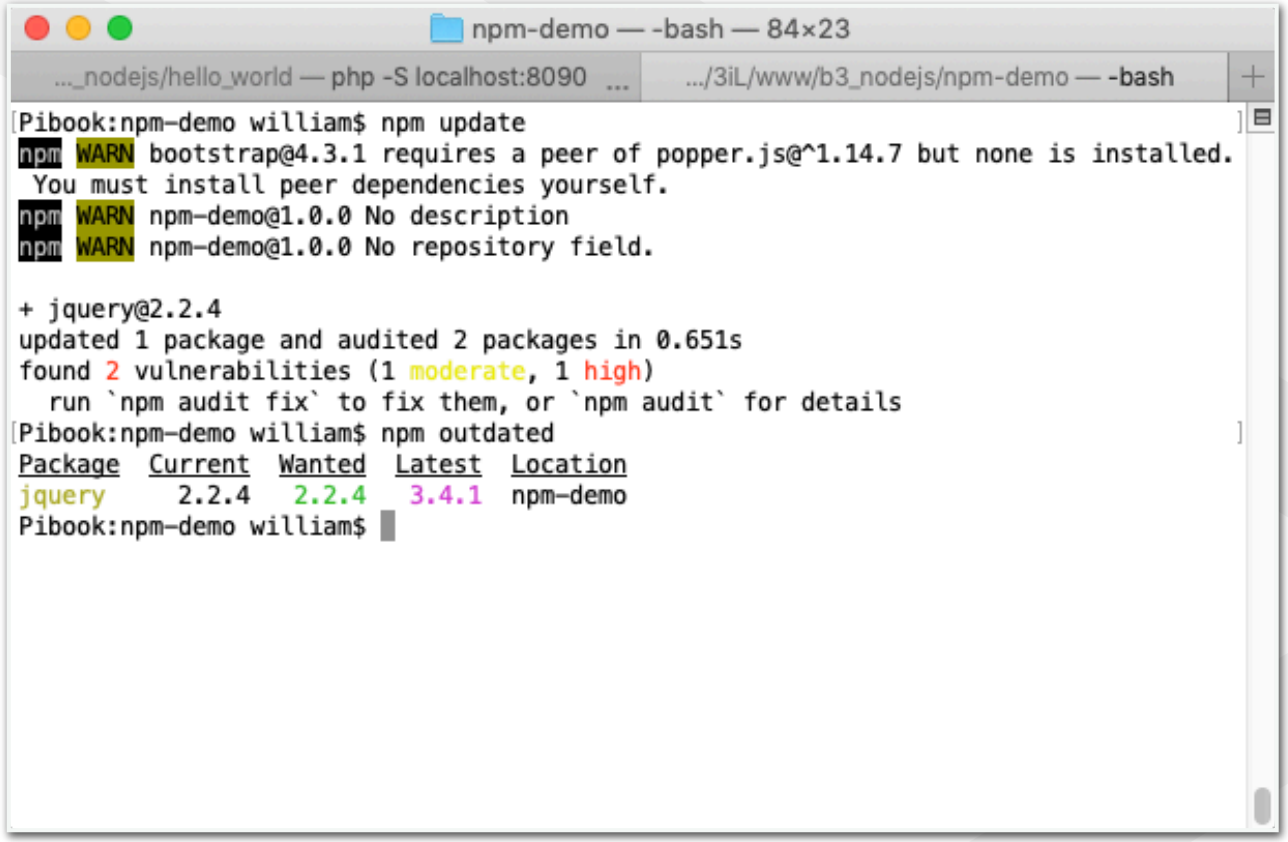
- Installer la version 2.1.1 de jQuery
- Pour lister les dépendances qui peuvent être mises à jour :

```
npm outdated
```



METTRE À JOUR LES DÉPENDANCES

- Taper `npm update` :



DÉPENDANCES DE DÉVELOPPEMENT

- Certaines dépendances peuvent ne concerner que la phase de développement :
  - Aucun intérêt à être présentes en production
  - Principe : tout ce qui n'est pas utile peut être dangereux
  - Exemple : tout ce qui est relatif aux tests ou à faciliter la vie lors du développement

- Exemple : `nodemon`
  - Package servant à relancer automatiquement l'application en cas de modification de ses sources
  - N'a pas de raison d'être pour la version en production de l'application

- npm permet de séparer ce type de dépendances en ajoutant `--save-dev` lors de leur installation

DÉPENDANCES DE DÉVELOPPEMENT

- Taper : `npm install nodemon --save-dev`
- Observer le contenu du fichier `package.json`

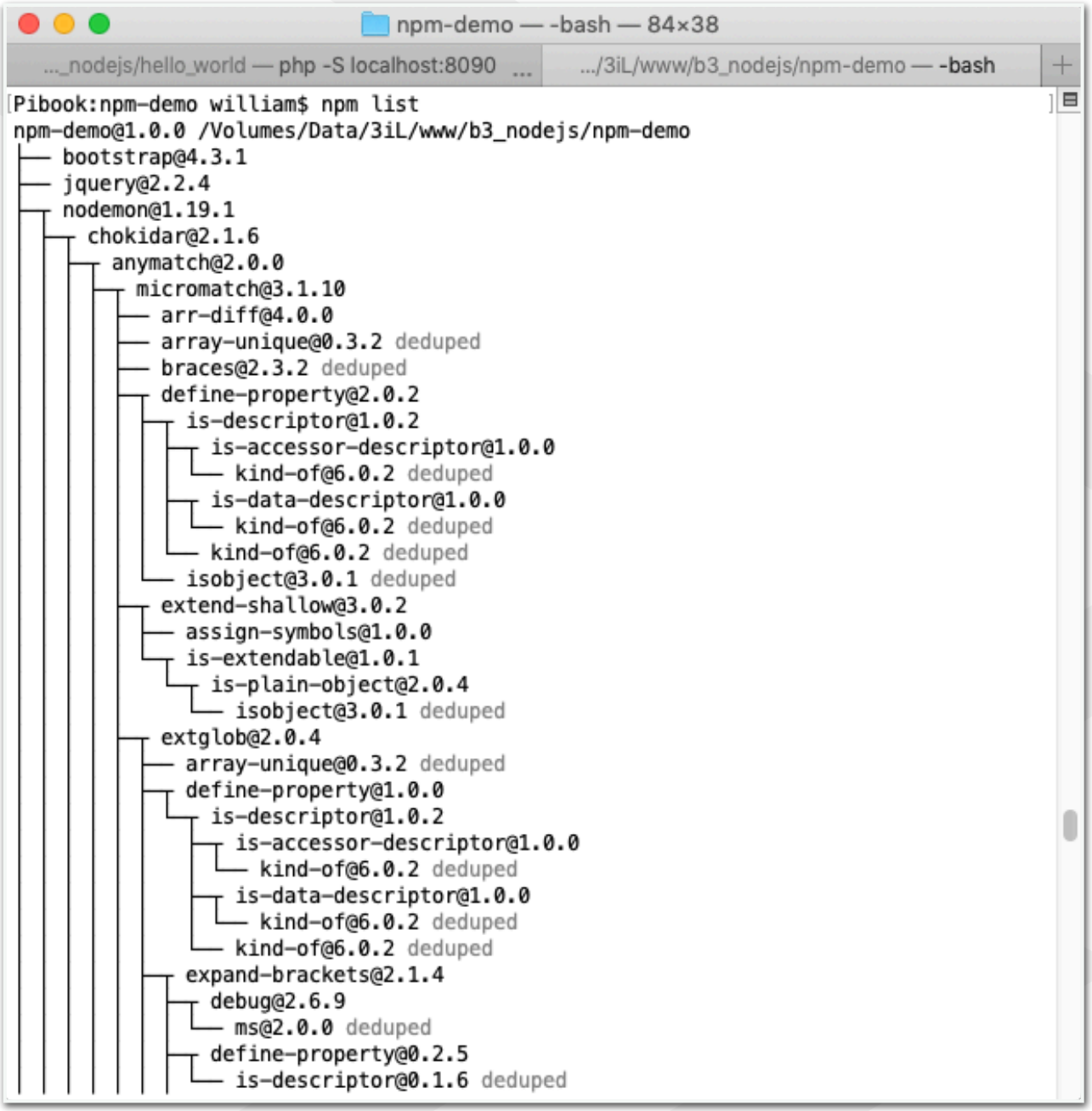
package.json :

```
1  {
2    "name": "npm-demo",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "bootstrap": "^4.3.1",
13     "jquery": "^2.2.4"
14   },
15   "devDependencies": {
16     "nodemon": "^1.19.1"
17   }
18 }
19
```

liste des dépendances du développement du projet

DÉPENDANCES DE DÉVELOPPEMENT

- Au passage, faire un `npm list` pour se rendre compte que la liste s'est étoffée :



npm
Partie 3

UTILISER NODEMON

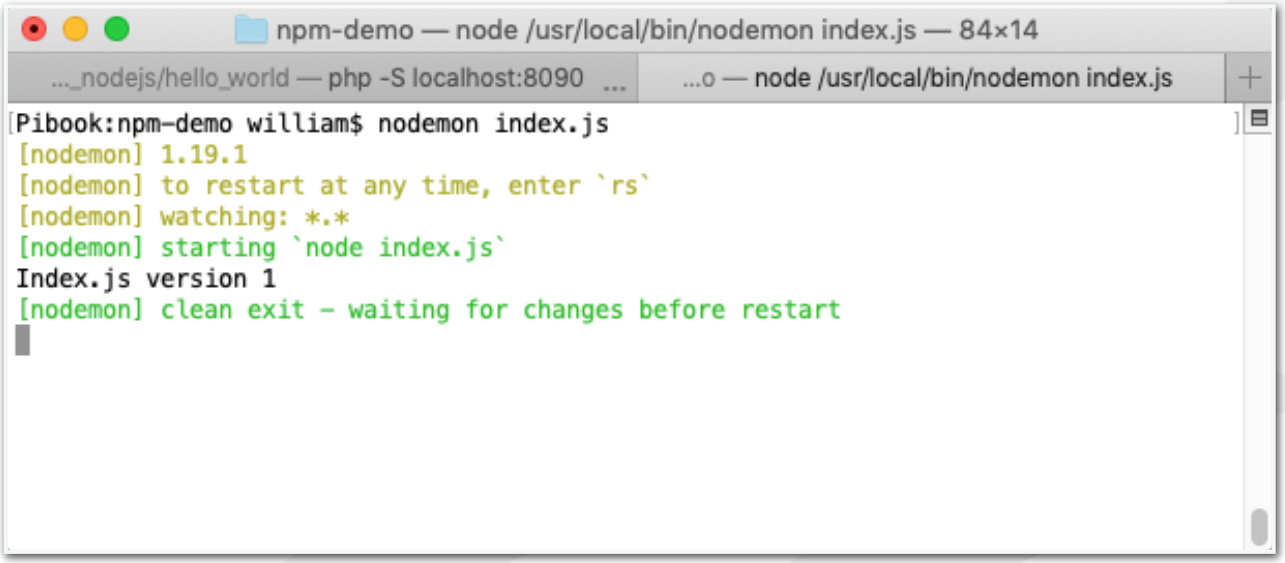
- Pour utiliser nodemon, nous allons :
  - Créer un fichier `index.js` simple (affichage)
  - Démarrer le projet avec nodemon
  - Modifier le fichier index.js

- Créer le fichier `index.js` comme suit :

package.json :

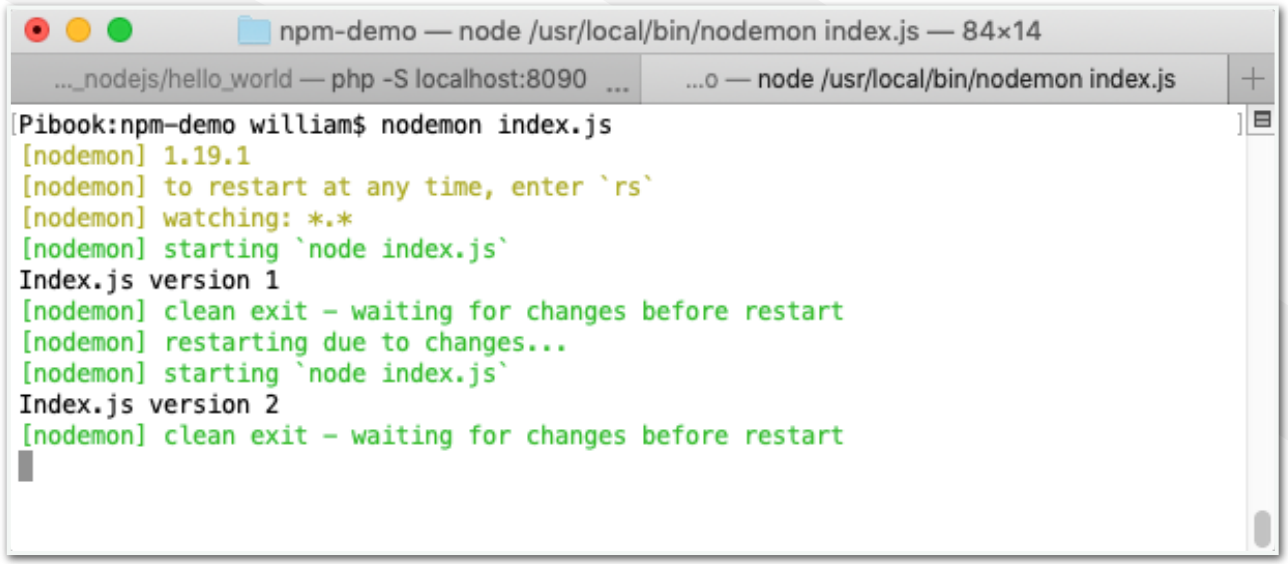
```
1   console.log("Index.js version 1");
```

- Taper `nodemon index.js`



UTILISER NODEMON

- Modifier le fichier `index.js` pour passer "à la version 2".
- Automatiquement `index.js` sera relancé dans le terminal.



- Nodemon scan en permanence les modifications dans le dossier.
- Même la modification d'un fichier qui n'intervient pas directement provoquera la relance de `index.js`.
- C'est particulièrement pratique pour relancer un programme serveur automatiquement.

UTILISER LA RUBRIQUE SCRIPTS

- La rubrique `scripts` de `package.json` peut être enrichie de nos propres scripts.

package.json :

```
{
  "name": "npm-demo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start":"nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bootstrap": "^4.3.1",
    "jquery": "^2.2.4"
  },
  "devDependencies": {
    "nodemon": "^1.19.1"
  }
}
```

- Il suffit maintenant de faire `npm run start` au lieu de `nodemon index.js`
- Comme `start` est souvent utilisée, on peut même taper `npm start` (cas particulier)

INSTALLATION GLOBALE

- Par défaut, npm réalise l'installation dans le dossier local.
- Possibilité de faire une installation "globale" (centralisée)
  - Utile surtout pour des outils comme Nodemon, LESS ou SASS
  - Permet d'enrichir le terminal de nouvelles commandes disponibles de partout
  - N'a pas forcément de sens pour tous les packages

- Pour faire une installation globale de LESS

```
npm install -g less
```

- Ne modifie pas le fichier `package.json`

- Attention : il faut avoir les droits pour que cela fonctionne.
- La commande `less` doit être disponible dans le terminal.

```
less --v
```