npm是一个node包管理和分发工具，已经成为了非官方的发布node模块（包）的标准。有了npm，可以很快的找到特定服务要使用的包，进行下载、安装以及管理已经安装的包。

1、npm install moduleNames：安装Node模块  
安装完毕后会产生一个node\_modules目录，其目录下就是安装的各个node模块。

node的安装分为全局模式和本地模式。  
一般情况下会以本地模式运行，包会被安装到和你的应用程序代码的本地node\_modules目录下。  
在全局模式下，Node包会被安装到Node的安装目录下的node\_modules下。

全局安装命令为$npm install -g moduleName。  
获知使用$npm set global=true来设定安装模式，$npm get global可以查看当前使用的安装模式。

示例：  
npm install express   
默认会安装express的最新版本，也可以通过在后面加版本号的方式安装指定版本，如npm install express@3.0.6

npm install <name> -g   
将包安装到全局环境中

但是代码中，直接通过require()的方式是没有办法调用全局安装的包的。全局的安装是供命令行使用的，就好像全局安装了vmarket后，就可以在命令行中直接运行vm命令

npm install <name> --save   
安装的同时，将信息写入package.json中项目路径中如果有package.json文件时，直接使用npm install方法就可以根据dependencies配置安装所有的依赖包，这样代码提交到github时，就不用提交node\_modules这个文件夹了。

2、npm view moduleNames：查看node模块的package.json文件夹  
注意事项：如果想要查看package.json文件夹下某个标签的内容，可以使用$npm view moduleName labelName

3、npm list：查看当前目录下已安装的node包  
注意事项：Node模块搜索是从代码执行的当前目录开始的，搜索结果取决于当前使用的目录中的node\_modules下的内容。$ npm list parseable=true可以目录的形式来展现当前安装的所有node包

4、npm help：查看帮助命令

5、npm view moudleName dependencies：查看包的依赖关系

6、npm view moduleName repository.url：查看包的源文件地址

7、npm view moduleName engines：查看包所依赖的Node的版本

8、npm help folders：查看npm使用的所有文件夹

9、npm rebuild moduleName：用于更改包内容后进行重建

10、npm outdated：检查包是否已经过时，此命令会列出所有已经过时的包，可以及时进行包的更新

11、npm update moduleName：更新node模块

12、npm uninstall moudleName：卸载node模块

13、一个npm包是包含了package.json的文件夹，package.json描述了这个文件夹的结构。访问npm的json文件夹的方法如下：  
$ npm help json   
此命令会以默认的方式打开一个网页，如果更改了默认打开程序则可能不会以网页的形式打开。

14、发布一个npm包的时候，需要检验某个包名是否已存在  
$ npm search packageName

15、npm init：会引导你创建一个package.json文件，包括名称、版本、作者这些信息等

16、npm root：查看当前包的安装路径  
npm root -g：查看全局的包的安装路径

17、npm -v：查看npm安装的版本

npm介绍

npm全称为Node Package Manager，是一个基于Node.js的包管理器，也是整个Node.js社区最流行、支持的第三方模块最多的包管理器。至今，npm已经管理约24万个由开发者、公司和社区创建的模块。

npm用法

npm的使用很简单，记住以下命令即可很好地使用它。

* npm init   
  运行构建新项目的向导
* npm install module\_name   
  在项目中安装一个模块
* npm install -g module\_name   
  全局方式安装一个模块
* npm install module\_name –save   
  在项目中安装一个模块，并把此模块添加到项目配置文件package.json中，作为项目依赖
* npm install module\_name –save-dev   
  在项目中安装一个模块，并把此模块添加到项目配置文件package.json中，作为项目开发依赖（devDependency）
* npm list   
  列出项目中已安装的所有模块
* npm list -g   
  列出系统中全局安装的所有模块
* npm remove module\_name   
  从项目中移除已安装的模块
* npm remove -g module\_name   
  从系统的全局安装中移除已安装的模块
* npm remove module\_name –save   
  从项目中移除已安装的模块，并从配置依赖中移除依赖关系
* npm remove module\_name –save-dev   
  从项目中移除已安装的模块，并从配置依赖中移除开发依赖（devDependency）关系
* npm update module\_name   
  更新指定的已安装模块的版本
* npm update -g module\_name   
  更新指定的全局安装模块的版本
* npm -v   
  显示npm包管理器的当前版本
* npm adduser username   
  在npmjs.org创建一个账户
* npm whoami   
  显示你在npmjs.org上的账户详细信息
* npm publish   
  发布自己开发的模块到npmjs.org，要发布模块必须先有账户

package.json说明

什么是Node.js的模块（Module）？在Node.js中，模块是一个库或框架，也是一个Node.js项目。Node.js项目遵循模块化的架构，当我们创建了一个Node.js项目，意味着创建了一个模块，这个模块的描述符文件，被称为package.json。

package.json内容出错，会导致项目出现bug，甚至阻止项目的运行。一个简单的package.json，其基本结构如下：

{

"name": "test-project",

"version": "1.0.0",

"description": "test project",

"private": true,

"main": "app.js",

"scripts": {

"test": "node test.js",

"start": "node app.js",

"clean": "rm -rf node\_modules"

},

"author": "lq",

"license": "ISC",

"dependencies": {

"module-1": "\*"

},

"devDependencies": {

"module-2": "\*"

}

}

它是一个合法的JSON数据，其含义一目了然。   
要注意author参数，它的值是你在[https://npmjs.org](https://npmjs.org/)网站的有效账户名，遵循“账户名<邮件>”这样的格式，比如”User [user@email.com](mailto:user@email.com)”。

Node.js模块的版本级别

Node.js模块有三种版本级别：

* 主要版本Major：X.0.0
* 次要版本Minor：0.X.0
* 补丁版本Patch：0.0.X

开发时应遵循这个约定。

npm的任务自动化

package.json中的scripts定义了一些任务，比如：

"scripts": {

"test": "node test.js",

"start": "node app.js",

"clean": "rm -rf node\_modules"

},

这个配置这定义了三个任务脚本：启动start、测试test、清理clean。要执行脚本，分别使用命令：

$ npm run start

$ npm run test

$ npm run clean

即可执行。