

[Udemy] npm - Mastering the Basics

Seção: 1 Mastering the Basics

1. Introduction

- Package Manager (Gerenciador de pacotes)

2. Getting npm

- [Linux] - Instalação do NodeJS

3. npm Help

- Ajuda sobre o NPM
- Ajuda rápida no <comand>
- Buscar por um termo na documentação
- Buscar ajuda para um command

4. package.json

- Dois principais benefícios do package.json
- Incluir, nos projetos, um arquivo chamado package.json

5. package.json defaults

- Definir valores defaults (autor, licença, etc.)
- Verificar os valores defaults definidos
- Apagar um valor default

6. Installing Local Packages

7. Uninstalling Local Packages

- Para desinstalar um pacote como dependência
- Para remover a dependência e o registro em package.json, usar a flag --save
- Para remover a dependência dev e o registro em package.json, usar a flag --save-dev

8. Installing Global Packages

9. Uninstalling Global Packages

- Comandos similares (com o mesmo resultado)

10. Listing Packages

- Listar os pacotes instalados local e globalmente e suas dependências, numa estrutura de árvore
- Para restringir a profundidade da árvore exibida, usar a flag --depth
- Listar packages globais

11. npm Versioning

- Instalar uma versão específica do pacote (em vez da versão mais recente)
- Instalar uma versão principal e secundária específica, mas com a versão do patch mais recente
- Instalar uma versão principal específica, mas com a versão secundária e do patch mais recente

12. Installing from package.json

- Especificar a versão no package.json

13. Updating Packages

- Atualizar para a versão mais recente
- Atualizar a dependência dev para a versão mais recente
- Atualizar todas as dependências e as dependências dev
- Atualizar pacotes globais
- Atualizar o próprio NPM para a versão mais recente (necessário privilégios de administrador)

- 14. npm Prune
- 15. Shortcuts
- 16. npm Scripts

Seção: 1 Mastering the Basics

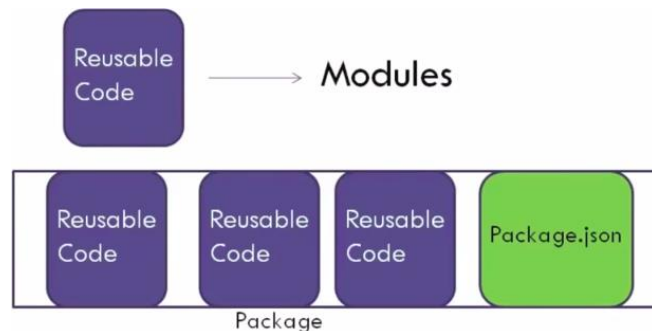
1. Introduction

npm, tudo em minúsculo, é o gerenciador de pacotes para Javascript.

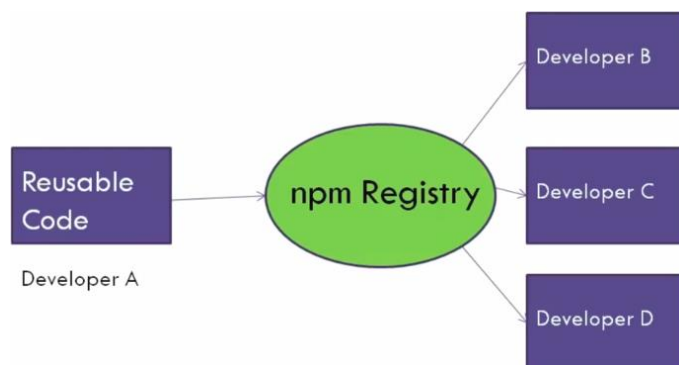
➤ Package Manager (Gerenciador de pacotes)

- Problem => Code
- Same problem => Reuse the Code

Arquivos individuais contendo código reusável são chamados módulos e um pacote nada mais é do que um diretório com um ou mais módulos junto com um arquivo especial, denominado package.json. Esse arquivo possui os metadados sobre o pacote.



- Você escreveu um código javascript para resolver um problema particular [Developer A]
- NPM permite que você publique seu código no registro do NPM para que outros desenvolvedores que estão enfrentando o mesmo problema possam reusar este código
- e quando você faz atualizações no seu código, fica mais fácil para os desenvolvedores dependentes dele checar estas atualizações e fazer o download delas.
- Basicamente, o npm é uma maneira de compartilhar seu código com outros desenvolvedores, reutilizar o código de outros desenvolvedores e gerenciar facilmente as diferentes versões do seu código



2. Getting npm

- NPM é fornecido como node.js
- Baixar e instalar o node.js em <https://nodejs.org/> (o lançamento mais recente e estável)
- Para verificar a versão do NodeJS instalada

```
node -v
```

- Para verificar a versão do npm instalada

```
npm -v
```



- Como Instalar Utilizando o NVM

Nesse artigo, iremos usar uma ferramenta chamada nvm, que significa "Node.js version manager" ou "Gerenciador de Versão do Node.js" para realizar a instalação do NodeJS e do NPM.

Usando o nvm, você poderá controlar o seu ambiente de desenvolvimento mais facilmente, com ele teremos acesso a uma série de versões da plataforma NodeJS de tal modo a possibilitar a instalação da que for mais apropriada para as nossas necessidades.

Para começar, precisaremos obter os pacotes de software do nosso repositório Ubuntu que nos permitirão compilar pacotes de fontes. O script `nvm` aproveitará estas ferramentas para construir os componentes necessários. Para começar, vamos abrir o Terminal do Linux, em seguida vamos rodar os seguintes comandos no Terminal:

```
sudo apt-get update
sudo apt-get install build-essential libssl-dev
```

Segue abaixo as imagens da execução dos comandos acima concluídos.

```

super@superpc:~$ sudo apt-get update
[sudo] senha para super:
Atualizando http://br.archive.ubuntu.com/ubuntu xenial InRelease
Get:1 http://br.archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:2 http://br.archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:3 http://br.archive.ubuntu.com/ubuntu xenial-updates/main amd64 Packages [549 kB]
Get:4 http://br.archive.ubuntu.com/ubuntu xenial-updates/main i386 Packages [549 kB]
Get:5 http://br.archive.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:6 http://br.archive.ubuntu.com/ubuntu xenial-updates/main Translation-en [238 kB]
Get:7 http://br.archive.ubuntu.com/ubuntu xenial-updates/main DEP-11 Metadata [372 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [794 kB]
Get:9 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 Packages [491 kB]
Get:10 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe i386 Packages [491 kB]
Get:11 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe Translation-en [194 kB]
Get:12 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe DEP-11 Metadata [163 kB]
Get:13 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe Translation-en [194 kB]
Get:14 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [794 kB]
Get:15 http://security.ubuntu.com/ubuntu xenial-security/main i386 Packages [794 kB]
Get:16 http://br.archive.ubuntu.com/ubuntu xenial-updates/universe DEP-11 Metadata [372 kB]
Get:17 http://br.archive.ubuntu.com/ubuntu xenial-backports/main amd64 Packages [32.0 kB]
Get:18 http://br.archive.ubuntu.com/ubuntu xenial-backports/main i386 Packages [32.0 kB]
Get:19 http://br.archive.ubuntu.com/ubuntu xenial-backports/main DEP-11 Metadata [32.0 kB]
Get:20 http://security.ubuntu.com/ubuntu xenial-security/main Translation-en [238 kB]
Get:21 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 Metadata [372 kB]
Get:22 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 Metadata Icons [47,3 kB]
Get:23 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [320,8 kB]
Get:24 http://security.ubuntu.com/ubuntu xenial-security/universe i386 Packages [320,8 kB]
Get:25 http://security.ubuntu.com/ubuntu xenial-security/universe Translation-en [320,8 kB]
Get:26 http://security.ubuntu.com/ubuntu xenial-security/universe DEP-11 Metadata [30,8 kB]
Get:27 http://security.ubuntu.com/ubuntu xenial-security/universe DEP-11 Metadata Icons [32,2 kB]
Retained 4.306 kB em 444 [111 KB/s]
Aplicando cache update completed, but some metadata was ignored due to errors.
Rebuild lista de pacotes... Pronto
super@superpc:~$

```

A terminal window titled "support@support-vm: ~" displays the output of several commands. The user runs "sudo apt-get install build-essential libssl-dev". The first part of the output shows the system checking package lists and building a dependency tree, confirming that build-essential is already installed. Then, it lists additional packages to be installed: libssl-dev, libssl1.0, and zlib1g-dev. A summary follows, indicating that three new packages will be installed and no existing ones will be removed or upgraded. Finally, it asks for confirmation to proceed with the installation, which has been answered affirmatively by typing "y".

```
support@support-vm:~$ sudo apt-get install build-essential libssl-dev
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informações de estado... Pronto
build-essential is already the newest version (12.1ubuntu2).
build-essential configured for automatic removal.
The following additional packages will be installed:
  libssl-dev libssl1.0 zlib1g-dev
Os novos pacotes a serem serão instalados:
  libssl-dev libssl1.0 zlib1g-dev
Os pacotes a serem serão atualizados:
  libssl1.0 zlib1g
3 pacotes a serem instalados, e 0 serão removidos e 0 não serão atualizados.
O espaço necessário é 3.722 kB de arquivos.
O espaço em disco disponível é 68,9 GB. O adicional de espaço em disco serão usados.
Você quer continuar? [S/n]
```

```
suporte@suporte:~$
The following additional packages will be installed:
libssl1.0.0 zlib1g-dev
Os novos pacotes a seguir serão instalados:
libssl-dev libssl-doc zlib1g-dev
Os pacotes a seguir serão atualizados:
libssl1.0.0 zlib1g
2 pacotes atualizados, 3 pacotes novos instalados, 0 a serem removidos e 404 não atualizados.
Preciso baixar 3.722 kB de arquivos.
Depois desta operação, 15,4 MB adicionais de espaço em disco serão usados.
Quer continuar? [y/n] y
Obter:1 http://br.archive.ubuntu.com/ubuntu/ xenial-updates/main amd64 zlib1g amd64 1:1.2.8.dfsg-2ubuntu1.1 [51,2 kB]
Obter:2 http://br.archive.ubuntu.com/ubuntu/ xenial-updates/main amd64 libssl1.0.0 amd64 1.0.2g-2ubuntu4.4 [1.081 kB]
Obter:3 http://br.archive.ubuntu.com/ubuntu/ xenial-updates/main amd64 libssl-dev amd64 1:1.2.8.dfsg-2ubuntu1.1 [168 kB]
Obter:4 http://br.archive.ubuntu.com/ubuntu/ xenial-updates/main amd64 libssl-doc amd64 1.0.2g-2ubuntu4.4 [1.341 kB]
Obter:5 http://br.archive.ubuntu.com/ubuntu/ xenial-updates/main amd64 libssl-doc all 1.0.2g-2ubuntu4.4 [1.350 kB]
Instalador 3.722 kB em 38s (354 kB/s)
Preparando pacotes...
(Levando banco de dados... 17601s /cheirira e diretórios atualmente instalados.)
A preparar para descompactar .../zlib1g_1:1.2.8.dfsg-2ubuntu1.1_amd64.deb ...
A descompactar zlib1g:amd64 (1:1.2.8.dfsg-2ubuntu1.1) sobre (1:1.2.8.dfsg-2ubuntu1) ...
A processar triggers para libc-bin (2.23-0ubuntu1) ...
Configurando zlib1g:amd64 (1:1.2.8.dfsg-2ubuntu1.1) ...
(Levando banco de dados... 17601s /cheirira e diretórios atualmente instalados.)
A preparar para descompactar .../libssl1.0.0_1.0.2g-2ubuntu4.4_amd64.deb ...
A descompactar libssl1.0.0:amd64 (1.0.2g-2ubuntu4.4) sobre (1.0.2g-2ubuntu1.1) ...
A selecionar pacote anteriormente não selecionado: zlib1g-dev:amd64.
A preparar para descompactar .../zlib1g-dev_1:1.2.8.dfsg-2ubuntu1.1_amd64.deb ...
A descompactar zlib1g-dev:amd64 (1:1.2.8.dfsg-2ubuntu1.1) sobre (1:1.2.8.dfsg-2ubuntu1) ...
A selecionar pacote anteriormente não selecionado: libssl-dev:amd64.
A descompactar libssl-dev:amd64 (1.0.2g-2ubuntu4.4) ...
A selecionar pacote anteriormente não selecionado: libssl-doc.
A preparar para descompactar .../libssl-doc_1.0.2g-2ubuntu4.4_all.deb ...
A descompactar libssl-doc (1.0.2g-2ubuntu4.4) ...
A processar triggers para libc-bin (2.23-0ubuntu1) ...
A processar triggers para libc-bin (2.23-0ubuntu1) ...
Configurando libssl1.0.0:amd64 (1.0.2g-2ubuntu4.4) ...
Configurando zlib1g-dev:amd64 (1:1.2.8.dfsg-2ubuntu1.1) ...
Configurando libssl-dev:amd64 (1.0.2g-2ubuntu4.4) ...
Configurando libssl-doc (1.0.2g-2ubuntu4.4) ...
A processar triggers para libc-bin (2.23-0ubuntu1) ...
suporte@suporte:~$
```

Uma vez que os pacotes requeridos estejam instalados, você poderá baixar o script de instalação do nvm com o curl.

É importante que antes da execução você mude o diretório no terminal para a Área de Trabalho (desktop), pois do contrário o script de instalação não será baixado, e, para mudar o diretório do Terminal, você pode usar o comando cd (change directory) e na sequência usar o comando abaixo para baixar o script de instalação:

```
curl -sL https://raw.githubusercontent.com/creationix/nvm/v0.31.0/install.sh -o install_nvm.sh
```

Obs.: O número da versão pode mudar com o tempo, então recomendo você acessar a página do projeto no GitHub (<https://github.com/creationix/nvm>) e procurar pela nova versão.

Na sequência, quando o terminal for liberado para digitação novamente, utilize o comando ls (list) para verificar se foi criado um arquivo install_nvm.sh na pasta em que você está.

Agora que o script de instalação foi criado, devemos executá-lo, e podemos fazer isso através do comando bash, ficando da seguinte maneira:

```
bash install_nvm.sh
```

Basta esperar a execução do comando finalizar e já temos o NVM instalado em nossa máquina.

Abaixo temos uma imagem com o processo descrito acima:

```
suporte@suporte:~/Área de Trabalho$
suporte@suporte:~$ cd /Área de Trabalho/
suporte@suporte:~/Área de Trabalho$ curl -sL https://raw.githubusercontent.com/creationix/nvm/v0.31.0/install.sh -o install_nvm.sh
suporte@suporte:~/Área de Trabalho$ ls
install_nvm.sh
suporte@suporte:~/Área de Trabalho$ bash install_nvm.sh
suporte@suporte:~/Área de Trabalho$
suporte@suporte:~$
suporte@suporte:~$ cd /home/suporte/.nvm/
suporte@suporte:~$
suporte@suporte:~$
```

Agora para podermos acessar as funções do NVM, temos que reiniciar o Terminal, é só fechá-lo e abri-lo novamente. Feito isso podemos instalar o NodeJS.

Se você usar o comando:

```
nvm ls-remote
```

Será listado no terminal todas as versões disponíveis do NodeJS, como mostra a imagem abaixo:



No momento da escrita desse artigo a última versão disponível era a v8.1.2.

Decidimos instalar a versão 6.11.0 do NodeJS por ser a versão mais estável no momento, para realizar a instalação basta você usar o comando:

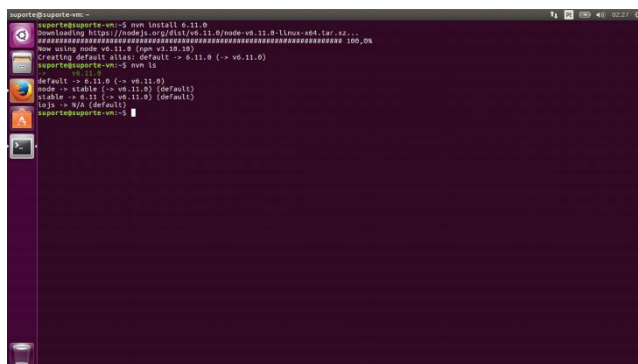
```
nvm install 6.11.0
```

Assim que a execução do comando terminar você já terá o NodeJS na versão 6.11.0 e o NPM 3.10.10 instalados na sua máquina. Para listar as versões do NodeJS instaladas através do NVM basta usar o comando:

```
nvm ls
```

Esse comando deverá listar apenas a versão 6.11.0 e mostra-la como a versão padrão.

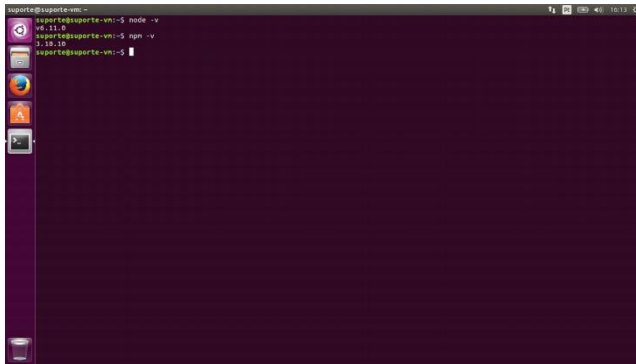
A imagem abaixo mostra o processo de instalação do NodeJS, através do NVM, e a listagem das versões instaladas:



E pronto! Você já tem o NVM, o NodeJS e o NPM instalados em sua máquina. Para garantir que o NodeJS e o NPM estejam funcionando você pode fazer o teste com os seguintes comandos:

```
node -v  
npm -v
```

Os comandos acima deverão ter como resultado algo como a imagem abaixo:



3. npm Help

➤ Ajuda sobre o NPM

```
npm help
```

➤ Ajuda rápida no <comand>

```
npm <command> -h
```

Exemplo: `npm install -h`

➤ Buscar por um termo na documentação

Enumera todas as ocorrências do <term> na documentação e os comandos de ajuda que contêm informações sobre o referido <term>

```
npm help-search <term>
```

Exemplo: `npm help-search update`

➤ Buscar ajuda para um command

Após a execução do comando acima, pode-se executar o comando abaixo para obtenção de informações mais detalhadas.

```
npm help <command>
```

4. package.json

- Manage dependencies: Express and its version
- Scripts: Initial build

➤ Dois principais benefícios do package.json

- Gerenciar as dependências do projeto
- Adicionar scripts que ajudam com o build inicial do projeto

➤ **Incluir, nos projetos, um arquivo chamado package.json**

- a. Criar um diretório (exemplo: npm)
- b. Abrir o terminal neste diretório e digitar

```
npm init
```

- c. Para cada opção exibida (e a possível resposta default), apertar <enter>
- d. Após todas as perguntas, é exibido uma prévia de como ficará o arquivo package.json. Tecle <enter> para criar o arquivo
- e. Ao abrir o arquivo, temos um nome, uma versão, uma descrição (que é o arquivo principal), alguns scripts, um autor e uma licença
- f. Caso deseje-se pular as perguntas iniciais, mantendo as respostas defaults, usar:

```
npm init --yes
```

5. package.json defaults

➤ **Definir valores defaults (autor, licença, etc.)**

- a. No terminal, escrever os comandos

```
npm config set init-author-name "<nome_autor>"
```

Substituir <nome_autor> por Vishwas (por exemplo).

```
npm set init-license "<license>"
```

Substituir <license> por MIT (por exemplo).

Obs.: A palavra config pode ser omitida.

- b. Criar o arquivo package.json e verificar os novos valores defaults de author e license

```
npm init -yes  
vi package.json
```

➤ **Verificar os valores defaults definidos**

- É informado o author definido

```
npm config get init-author-name
```

- É informado a license definida

```
npm get init-license
```

Obs.: A palavra config pode ser omitida.

➤ **Apagar um valor default**


```
npm config delete init-author-name
npm config delete init-license
```

6. Installing Local Packages

Quando você quer usar um pacote para seu projeto e esse pacote faz sentido apenas para esse específico projeto, você pode instalar o pacote localmente na pasta do projeto.

- a. Abrir o terminal no diretório do projeto
- b. Executar o comando

```
npm install <package-name>
```

- c. Após alguns segundos, será criado um diretório específico chamado node_modules. Dentro do qual, serão encontrados todos os pacotes instalados.
- d. Para que seja incluído a dependência ao <package-name> no arquivo package.json, deve-se executar o comando com a flag --save

```
npm install <package-name> --save
```

- Registro no package.json:

```
"dependencies": {
  "<package-name>": "^x.y.z"
}
```

- e. Para instalar pacotes apenas com o propósito de Desenvolvimento (para testes, por exemplo) e não de Produção, usar a flag -dev

```
npm install <package-name> --save-dev
```

- Registro no package.json:

```
"devDependencies": {
  "<package-name>": "^ x.y.z "
}
```

7. Uninstalling Local Packages

- Para desinstalar um pacote como dependência

```
npm uninstall <package-name>
```

Irá remover o <package-name>, mas não irá remover o registro em package.json

- Para remover a dependência e o registro em package.json, usar a flag --save

```
npm uninstall <package-name> --save
```

- Para remover a dependência dev e o registro em package.json, usar a flag --save-dev

```
npm uninstall <package-name> --save-dev
```

Obs.: A pasta node_modules será automaticamente excluída se todas as dependências forem desinstaladas.

8. Installing Global Packages

Agora alguns pacotes, como grunt ou gulp, precisam ser usados a partir da linha de comando e, em tais cenários, deve-se instalar o pacote globalmente. Para isso, deve-se usar a flag -g.

```
npm install <package-name> -g
```

O comando irá instalar a dependência, mas na pasta node_modules não será adicionado nenhum diretório e no arquivo package.json não será adicionado nenhuma dependência. Logo mais, é informado onde os pacotes globais instalados ficam.

9. Uninstalling Global Packages

```
npm uninstall <package-name> -g
```

- Comandos similares (com o mesmo resultado)

```
npm remove <package-name>
npm rm <package-name>
npm un <package-name>
```

10. Listing Packages

- Listar os pacotes instalados local e globalmente e suas dependências, numa estrutura de árvore

```
npm list
```

- Para restringir a profundidade da árvore exibida, usar a flag --depth

```
npm list --depth <number>
```

- 0 (zero) irá listar apenas os packages do projeto, sem nenhuma dependência
- 1 (um) irá listar todos os packages com as suas dependências imediatas

- Listar packages globais

```
npm list --global true --depth 0
```

Os packages globais ficam localizados no diretório oculto exibido após a execução do comando acima, dentro do diretório node_modules.

```
wjuniori@inspiron-5548:~/git/npm$ npm list --global true --depth 0
```

```
/home/wjuniori/.nvm/versions/node/v8.11.3/lib
└─ moment@2.22.2
└─ npm@5.6.0
```

11. npm Versioning

- É muito importante entender como o versionamento (ou controle de versão) funciona com os pacotes.
- Versão semântica é uma especificação onde uma versão é representada por três números que significam a mesma coisa para qualquer desenvolvedor.
- Então, ao usar `npm install`, um pacote por padrão é instalado com a última versão estável representada por três números.

```
npm install <package-name> --save
```

Registro no `package.json`:

```
"dependencies": {
  "<package-name>": "^x.y.z"
}
```

O primeiro número representa a versão principal, o segundo representa a versão secundária e o terceiro representa a versão do patch.

Sempre que houver uma correção de bug ou uma melhoria de desempenho, a versão do patch é incrementada. Já a versão secundária é incrementada quando há uma nova feature (recurso), mas essa feature não quebra nenhuma funcionalidade já existente. E finalmente a versão principal é incrementada somente quando há uma quebra, ou seja, uma interrupção na funcionalidade existente. Por exemplo, em Angular I e II houveram muitas mudanças que quebraram a funcionalidade existente e então a versão principal foi atualizada.

➤ Instalar uma versão específica do pacote (em vez da versão mais recente)

Usar o símbolo `@` seguido da versão

```
npm install <package-name>@x.y.z --save
```

Após a instalação, no arquivo `package.json`, a versão do pacote dependente será atualizada para a versão especificada no comando.

➤ Instalar uma versão principal e secundária específica, mas com a versão do patch mais recente

```
npm install <package-name>@x.y --save
```

- Registro no `package.json`:

```
"dependencies": {
  "<package-name>": "^x.y.12"
}
```

```
}
```

A versão do patch 12 é a mais recente para versão principal e secundária x.y.

- **Instalar uma versão principal específica, mas com a versão secundária e do patch mais recente**

```
npm install <package-name>@x --save
```

- Registro no package.json:

```
"dependencies": {  
  "<package-name>": "^x.17.10"  
}
```

Então, para a versão principal x, a versão secundária 17 e a versão do patch 10 são as mais recentes.

12. Installing from package.json

O arquivo readme de qualquer projeto sempre especifica que primeiro deve-se executar o comando npm install. Este é o uso típico porque, ao executar npm install, todos os pacotes listados como dependências no package.json são instalados no diretório local do projeto.

- a. Digamos que o projeto npm depende de <package-name>. Isto está registrado no package.json
- b. Abrir o terminal no diretório do projeto e executar comando:

```
npm install
```

- c. Será criado o diretório 'node_modules' dentro do diretório do projeto
- d. Dentro de 'node_modules', será criado um diretório para cada package. Aqui onde os arquivos do package ficam.

- **Especificar a versão no package.json**

- Versão dos packages com ^ (acento circunflexo)

```
"dependencies": {  
  "<package-name>": "^4.14.1"  
}
```

O tipo mais comum é a versão precedida de ^ (acento circunflexo). Isto significa, ao executar npm install, manter a versão principal (neste caso, 4), mas instalar/recuperar a versão secundária e do patch mais recente para a versão principal definida (neste momento, <package-name>@4.17.10).

- Versão dos packages com ~ (til)

```
"dependencies": {  
  "<package-name>": "~4.14.1"  
}
```

A versão precedida de ~ (til) significa, ao executar npm install, manter a versão principal e a secundária (neste caso, 4.14), mas instalar/recuperar a versão do patch mais recente para a versão principal e secundária definida (neste momento, <package-name>@4.14.2).

- Versão dos packages específica

```
"dependencies": {  
  "<package-name>": "4.14.1"  
}
```

A versão sem qualquer precedência significa, ao executar npm install, manter a versão principal, a secundária e a do patch (neste caso, 4.14.1), mesmo que haja uma versão mais recente/superior.

- Versão dos packages mais recentes

```
"dependencies": {  
  "<package-name>": "*"  
}
```

A versão somente com asterisco (*) significa, ao executar npm install, instalar/recuperar a versão mais recente do package, independentemente da versão principal, secundária e do patch (neste momento, <package-name>@5.17.10).

13. Updating Packages

É essencial atualizar os pacotes de tempos em tempos para que se possa fazer uso dos novos recursos que estão disponíveis a cada lançamento.

➤ Atualizar para a versão mais recente

```
npm update <package-name> --save
```

➤ Atualizar a dependência dev para a versão mais recente

- Especificar uma dependência dev

```
npm update <package-name> --dev --save-dev
```

- Todas as dependências dev

```
npm update --dev --save-dev
```

➤ Atualizar todas as dependências e as dependências dev

```
npm update
```

➤ Atualizar pacotes globais

- Todos os pacotes globais

```
npm update -g
```

- Um pacote global específico

```
npm update -g <package-name>
```

➤ Atualizar o próprio NPM para a versão mais recente (necessário privilégios de administrador)

```
npm install npm@latest -g
```

14. npm Prune

Às vezes, podem ter sido instalados pacotes para realizar testes/experimentos ou que realmente não eram necessários. Então, estes pacotes estão no diretório `node_modules`, mas não estão mais listados no `package.json`. Ao executar o comando `npm list`, estas dependências não-listadas são informadas como `extraneous` (estranhas), o que significa dizer que os pacotes estão apenas no diretório do projeto, mas não estão no `package.json`.

- Remover os pacotes `extraneous` (estranhos) do diretório do projeto

```
npm prune
```

15. Shortcuts

Commands	Shortcuts	Description
<code>npm init --yes</code>	<code>npm init -y</code>	Criar um package como os valores defaults
<code>npm install <package-name></code>	<code>npm i <package-name></code>	Instalar um pacote localmente
<code>npm install <package-name> --save</code>	<code>npm i <package-name> -S</code>	Instalar um pacote localmente e adicioná-lo em <code>package.json</code>
<code>npm install <package-name> --save-dev</code>	<code>npm i <package-name> -D</code>	Instalar um pacote localmente como uma dependência dev
<code>npm --version</code>	<code>npm -v</code>	Informar a versão do NPM instalada
<code>--global</code>	<code>-g</code>	Escopo global

- Notações abreviadas para as diferentes opções:
<https://docs.npmjs.com/misc/config#shorthands-and-other-cli-niceties>

16. npm Scripts

- O npm contém um script de teste simples que apenas ecoa no console que nenhum teste foi especificado (ver package.json).
- Então, vamos criar um script simples que vai iniciar um aplicativo de carregamento simples (simple load application).

a. Criar um arquivo app.js (aplicativo simples de impressão no console)

//conteúdo do app.js

```
console.log('Npm Script Test');
```

b. Executar o app.js no terminal (imprime no console 'Npm Script Test')

```
node app.js
```

c. Em package.json, adicionar em "scripts" uma nova entrada separa por vírgula:

```
"start": "node app.js"
```

d. Executar no terminal

```
npm start
```

executará o app.js, exibindo a mensagem no console.

- Agora isso é muito útil quando você está desenvolvendo projetos. Digamos que você criou um projeto e o start para o projeto está no package.json ("node app.js"). Então, alguém faz o download do seu projeto e executa-o:

```
npm start
```

Mas depois você decide que o "start" será "node index.js". O usuário não sabe disso e, quando ele receber a versão mais recente e executar "npm start", vai funcionar da mesma forma. Então, por ter um comando de partida, o usuário não precisa saber o comando que será executado.

Quando você faz alterações, tudo o que os usuários precisam se preocupar é a execução do "npm start". E isso, por sua vez, vai executar o comando apropriado para iniciar o aplicativo.