Node Package Manager

NPM, acronimo di Node Package Manager, è il gestore di pacchetti predefinito per Node.js, un ambiente di runtime JavaScript. Node.js è ampiamente utilizzato per sviluppare applicazioni lato server e per lo sviluppo di strumenti e utilità di linea di comando.

Alcune caratteristiche e concetti importanti relativi a NPM:

1. **Gestione dei pacchetti**: NPM facilita il processo di installazione, aggiornamento e rimozione dei pacchetti JavaScript. I pacchetti possono includere librerie, framework, tool e altri componenti utili per lo sviluppo di applicazioni JavaScript.
2. **Pacchetti locali e globali**: NPM consente di installare pacchetti localmente all'interno di un progetto specifico o globalmente a livello di sistema. I pacchetti globali sono disponibili in tutto il sistema e possono essere utilizzati da qualsiasi progetto, mentre i pacchetti locali sono specifici per un progetto e vengono installati nella cartella del progetto.
3. **package.json**: Ogni progetto Node.js include un file `package.json`, che funge da manifesto per il progetto e contiene informazioni come il nome del progetto, la versione, le dipendenze (pacchetti utilizzati dal progetto) e altri metadati. Questo file può essere gestito manualmente o generato automaticamente utilizzando il comando `npm init`.
4. **Dipendenze**: NPM gestisce automaticamente le dipendenze dei pacchetti installati. Quando si installa un pacchetto, NPM verifica e installa anche tutte le dipendenze di quel pacchetto. Questo semplifica notevolmente la gestione delle dipendenze e assicura che tutte le dipendenze necessarie siano disponibili per il progetto.
5. **Scripts personalizzati**: NPM consente di definire script personalizzati nel file `package.json`, che possono essere eseguiti utilizzando il comando `npm run`. Questi script possono essere utilizzati per automatizzare attività comuni come la compilazione del codice, l'avvio del server di sviluppo o l'esecuzione di test.

6. **Registry pubblico**: NPM dispone di un vasto registro pubblico di pacchetti JavaScript disponibili per l'installazione. È possibile cercare e installare pacchetti dal registro pubblico utilizzando il comando `npm install`.

NPM Package

Un package npm è tipicamente una directory che contiene il file *`package.json`* , file che viene richiesto per pubblicare nel registry.

Per npm, un modulo è un qualsiasi file o directory nella cartella ***node_modules*** che può essere caricato tramite la funzione `require()` di node.js

Come creare un package

1. **Inizializza il progetto**: Assicurati di avere Node.js installato sul tuo sistema. Crea una nuova directory per il tuo progetto e accedi ad essa tramite il terminale. Poi, inizializza il tuo progetto npm eseguendo il comando:

   ```
   npm init
   ```

   Segui le istruzioni guidate per compilare le informazioni sul tuo pacchetto, come nome, versione, descrizione, autore, licenza, etc. Al termine, verrà generato un file `package.json`.

2. **Crea il codice del pacchetto**: Scrivi il codice del tuo pacchetto nel tuo editor preferito. Assicurati di includere tutte le funzionalità e i file necessari per il tuo pacchetto.

3. **Aggiungi file necessari**: Oltre ai file del codice sorgente, potresti voler includere altri file nel tuo pacchetto, come file di documentazione, file di licenza, file di esempio, ecc.

4. **Definisci il punto d'ingresso (entry point)**: Nel tuo file `package.json`, assicurati di specificare il file principale del tuo pacchetto, che verrà utilizzato come punto di ingresso quando qualcuno utilizzerà il tuo pacchetto. Puoi farlo impostando la chiave `main`.

5. **Testa il tuo pacchetto**: Assicurati che il tuo pacchetto funzioni correttamente eseguendo test appropriati. Puoi creare test utilizzando strumenti come Mocha, Jest, o qualsiasi altro framework di test che preferisci.
6. **Pubblica il tuo pacchetto (opzionale)**: Se desideri condividere il tuo pacchetto con altri sviluppatori, puoi pubblicarlo sul registro npm. Per fare ciò, devi creare un account npm e autenticarti dal tuo terminale utilizzando il comando `npm login`. Una volta autenticato, puoi pubblicare il tuo pacchetto eseguendo il comando `npm publish`.

Anatomia file package.json

```
"scripts": {
  "start": "node server.js",
  "test": "mocha test/*.js",
  "build": "webpack"
}
```

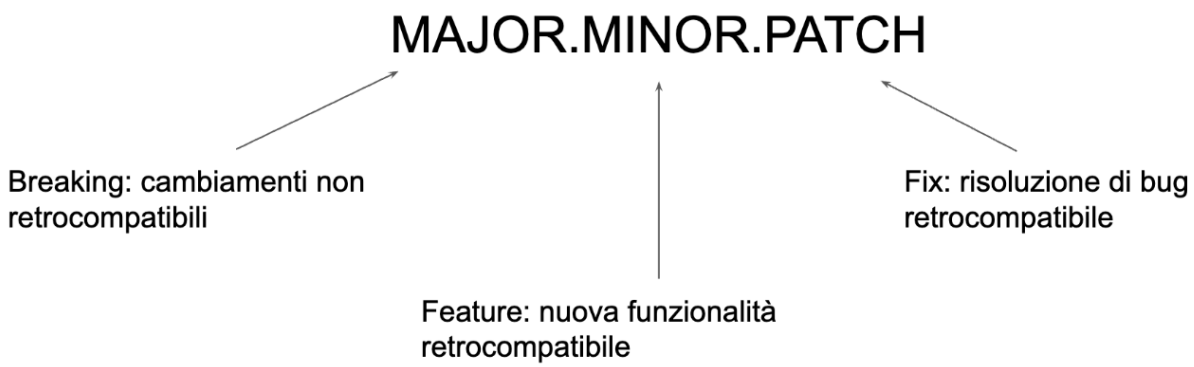
1. **name**: Il nome del pacchetto. Deve essere unico nel registro npm e può includere solo caratteri alfanumerici, trattini e underscore. Ad esempio: `"name": "il_mio_pacchetto"`.
2. **version**: La versione del pacchetto, seguendo il formato semantico di versionamento (Semantic Versioning). Questo è di solito composto da tre numeri separati da punti (es. 1.0.0), dove il primo è la versione principale, il secondo è la versione di sviluppo e il terzo è la versione di patch. Ad esempio: `"version": "1.0.0"`.
3. **description**: Una breve descrizione del pacchetto. È utile per gli utenti che cercano informazioni sul pacchetto. Ad esempio: `"description": "Un pacchetto che fa X e Y"`.
4. **main**: Il punto di ingresso principale del tuo pacchetto, ovvero il file JavaScript che sarà caricato quando qualcuno richiederà il tuo pacchetto. Ad esempio: `"main": "index.js"`.

5. **scripts**: Una sezione che contiene comandi personalizzati che possono essere eseguiti utilizzando il comando `npm run`.
6. **keywords**: Una serie di parole chiave che aiutano gli altri sviluppatori a trovare il tuo pacchetto quando cercano su npm. Ad esempio: `"keywords": ["node", "npm", "package"]`.
7. **author**: Il nome dell'autore del pacchetto. Ad esempio: `"author": "Nome Cognome <email@example.com>"`.
8. **license**: La licenza con cui il tuo pacchetto è distribuito. È importante specificare la licenza per informare gli utenti su come possono utilizzare il tuo codice. Ad esempio: `"license": "MIT"`.
9. **dependencies** e **devDependencies**: Le dipendenze del pacchetto. `dependencies` elenca i pacchetti richiesti per l'esecuzione del codice del pacchetto, mentre `devDependencies` elenca le dipendenze utilizzate solo durante lo sviluppo.

```
"dependencies": {
  "express": "^4.17.1",
  "lodash": "^4.17.21"
},
"devDependencies": {
  "mocha": "^9.1.3",
  "chai": "^4.3.4"
}
```

10. **repository**: L'URL del repository del codice sorgente del tuo pacchetto. Può essere utile per gli utenti interessati a contribuire o a esaminare il codice sorgente. Ad esempio: `"repository": "https://github.com/nomeutente/nome-repository.git"`.

Semantic Versioning



Il file `package-lock.json`

Questo file è generato automaticamente da npm per registrare l'albero delle dipendenze esatto di un progetto Node.js. È stato introdotto per garantire che le installazioni successive di pacchetti avvengano in modo deterministico e riproducibile su tutti i sistemi, eliminando eventuali differenze causate da aggiornamenti automatici dei pacchetti o da risoluzioni non deterministiche delle dipendenze.

1. **Albero delle dipendenze**: Il file `package-lock.json` contiene un albero delle dipendenze esatto per il progetto, incluso ogni pacchetto installato e le sue dipendenze, insieme alle versioni specifiche dei pacchetti. Questo assicura che le stesse versioni dei pacchetti vengano installate su tutti i sistemi e in tutti i momenti, garantendo la coerenza tra gli ambienti di sviluppo, di test e di produzione.
2. **Versioni esatte**: Ogni pacchetto nel file `package-lock.json` elenca una versione esatta, inclusi tutti i suoi submoduli e le loro versioni. Ciò elimina la possibilità di installare versioni diverse di un pacchetto su diversi sistemi, a meno che non vengano esplicitamente aggiornati.
3. **Integrità dei pacchetti**: Il file `package-lock.json` include anche un hash SHA-512 per ogni pacchetto installato, garantendo l'integrità dei pacchetti e delle loro dipendenze. Questo protegge da eventuali modifiche non autorizzate ai pacchetti durante il trasferimento o l'installazione.

4. **Esclusione dalla gestione del codice sorgente**: Il file `package-lock.json` è progettato per essere generato automaticamente da npm e non dovrebbe essere modificato manualmente. È incluso nel controllo del codice sorgente per garantire la coerenza tra gli sviluppatori, ma non è necessario modificarlo direttamente.

NPX

Strumento fornito con npm (Node Package Manager) che consente di eseguire pacchetti npm senza doverli installare globalmente o localmente. È particolarmente utile quando vuoi eseguire comandi da pacchetti specifici solo occasionalmente o quando non vuoi inquinare globalmente il tuo sistema con pacchetti che non utilizzerai spesso.

Alcune caratteristiche e utilizzi principali di `npx`:

1. **Esecuzione di comandi da pacchetti npm**: Con `npx`, puoi eseguire comandi direttamente da pacchetti npm senza preoccuparti di installarli globalmente o localmente. Ad esempio, se vuoi eseguire il comando `create-react-app` per creare un nuovo progetto React, puoi farlo senza installare `create-react-app` globalmente utilizzando `npx create-react-app my-app`.
2. **Esecuzione di versioni specifiche dei pacchetti**: `npx` consente di specificare una versione specifica di un pacchetto da utilizzare per eseguire il comando. Ad esempio, `npx -p eslint@7 eslint .` eseguirà il comando eslint utilizzando la versione 7 di eslint.
3. **Esecuzione di script in un progetto**: `npx` può essere utilizzato per eseguire script definiti nel file `package.json` di un progetto. Ad esempio, se hai uno script `lint` nel tuo `package.json`, puoi eseguirlo con `npx lint`.
4. **Esecuzione di comandi globali localmente**: `npx` può essere utilizzato per eseguire comandi globali localmente, senza doverli installare globalmente sul tuo sistema. Questo è utile se vuoi evitare la collisione di versione o se non hai i permessi per installare globalmente i pacchetti.
5. **Risoluzione delle dipendenze**: Prima di eseguire un comando con `npx`, verifica se il pacchetto richiesto è già installato localmente o se è disponibile

nella cache npm locale. Se non lo è, `npx` lo installerà temporaneamente nella cache locale prima di eseguire il comando.