NPM 快速参考

包管理、脚本、版本控制、发布、工作区

安装

安装 npm 与 Node

```
node -v && npm -v                    # check installed versions
npm install -g npm@latest            # update npm itself
nvm install --lts                    # install Node LTS via nvm
nvm use 20                           # switch to Node 20
```

安装命令

`npm install`	安装 package.json 中的所有依赖
`npm install pkg`	添加为依赖项
`npm install -D pkg`	添加为开发依赖
`npm install -g pkg`	全局安装包
`npm install pkg@2.1.0`	安装指定版本
`npm ci`	从 lock 文件全量安装（CI/CD）
`npm uninstall pkg`	移除包

包管理

管理包

```
npm ls                               # list installed packages
npm ls --depth=0                     # top-level only
npm outdated                         # check for newer versions
npm update                           # update within semver range
npm audit                            # check for vulnerabilities
```

管理命令

`npm ls`	以树形列出已安装的包
`npm outdated`	显示有新版本的包
`npm update [pkg]`	在 semver 范围内更新包
`npm audit`	审计依赖漏洞
`npm audit fix`	自动修复有漏洞的依赖
`npm prune`	移除多余的包
`npm dedupe`	扁平化依赖树以减少重复

脚本

运行脚本

```
npm run build                        # run "build" script
npm test                             # shortcut for "test" script
npm start                            # shortcut for "start" script
npm run lint -- --fix                # pass args to script
npm run dev &                        # run in background
```

脚本生命周期

`npm test / npm t`	运行 `scripts.test`
`npm start`	运行 `scripts.start`
`npm run <name>`	运行任意自定义脚本
`pre<name>`	在 `<name>` 前自动运行
`post<name>`	在 `<name>` 后自动运行
`npm run`	列出所有可用脚本

package.json

初始化与字段

```
npm init                             # interactive setup
npm init -y                          # accept all defaults
npm pkg set name="my-app"            # set a field
npm pkg get version                  # read a field
```

关键字段

`name`	包名（小写，无空格）
`version`	当前版本（semver：major.minor.patch）
`main`	CommonJS 入口点（`require`）
`module`	ES 模块入口点（打包工具）
`type`	`"module"` 表示 ESM，`"commonjs"` 表示 CJS（默认）
`scripts`	命名命令（build、test、start 等）
`dependencies`	生产依赖
`devDependencies`	仅开发时依赖
`engines`	所需 Node/npm 版本范围

版本管理

版本命令

```
npm version patch                    # 1.0.0 → 1.0.1
npm version minor                    # 1.0.1 → 1.1.0
npm version major                    # 1.1.0 → 2.0.0
npm version 3.2.1                    # set explicit version
npm version prerelease --preid=beta # 1.0.0-beta.0
```

Semver 范围

`^1.2.3`	兼容：>=1.2.3 <2.0.0（默认）
`~1.2.3`	补丁级：>=1.2.3 <1.3.0
`1.2.3`	仅精确版本
`>=1.0.0 <2.0.0`	显式范围
`*`	任意版本
`1.x / 1.2.x`	通配符范围
`latest`	最新发布的版本标签

发布

发布工作流

```
npm login                            # authenticate to registry
npm publish                          # publish public package
npm publish --access public          # scoped package as public
npm unpublish pkg@1.0.0              # remove specific version
npm deprecate pkg@"<2" "Use v2+"     # deprecate old versions
```

发布参考

`npm login`	向 npm 注册表认证
`npm publish`	将包发布到注册表
`npm pack`	创建 tarball 但不发布
`npm unpublish`	移除已发布版本（72 小时内）
`npm deprecate`	将版本标记为废弃
`.npmignore`	发布时排除的文件
`files (package.json)`	发布时包含的文件白名单

工作区

工作区命令

```
npm init -w packages/core            # create workspace
npm install -w packages/core lodash  # install in workspace
npm run build --workspaces           # run in all workspaces
npm run test -w packages/api         # run in specific workspace
npm ls --workspaces                  # list workspace deps
```

工作区配置

`workspaces (package.json)`	工作区 glob 数组：`["packages/*"]`
`-w / --workspace`	指定某个工作区
`--workspaces`	在所有工作区运行命令
`--include-workspace-root`	将根包包含在工作区操作中
`npm install (root)`	安装所有工作区依赖
`Hoisting`	共享依赖提升到根 node_modules

npx

使用 npx 运行

```
npx create-react-app my-app          # run without installing
npx tsc --init                       # run local or remote bin
npx -p typescript tsc file.ts        # specify package explicitly
npx --yes create-next-app            # skip install prompt
npx node@18 -e "console.log('hi')" # run with specific Node
```

npx 选项

`npx cmd`	从本地 node_modules/.bin 或远程运行命令
`npx -p pkg cmd`	安装 pkg，然后运行 cmd
`npx --yes cmd`	自动确认安装提示
`npx --no cmd`	拒绝安装——本地不存在则失败
`npx -c 'cmd'`	在 npx PATH 下运行 Shell 命令
`npx node@ver`	运行特定 Node.js 版本

配置

配置命令

```
npm config list                      # show current config
npm config set registry https://r.npmjs.com/
npm config set init-author-name "Name"
npm config get prefix                # global install path
npm config delete key                # remove a config value
```

配置参考

`.npmrc (project)`	项目级配置文件
`~/.npmrc`	用户级配置文件
`registry`	包注册表 URL
`save-exact`	`true` 表示安装时锁定精确版本
`engine-strict`	`true` 表示强制执行 `engines` 字段
`fund`	`false` 表示禁止显示资助消息
`audit`	`false` 表示安装时跳过审计

常见模式

单行命令

```
npm ls --depth=0 --json | jq '.dependencies | keys[]'
npm outdated --long                  # show type and homepage
npm cache clean --force              # clear npm cache
npm explain pkg                      # why is pkg installed?
npm exec -- envinfo --system         # system info for bug reports
```

实用技巧

`仅使用 lock 文件`	`npm ci`——从 package-lock.json 全量安装
`检查许可证`	`npx license-checker --summary`
`查找未使用的依赖`	`npx depcheck`
`包体积`	`npx bundlephobia-cli pkg`——检查包大小
`升级所有依赖`	`npx npm-check-updates -u && npm install`
`本地注册表`	`npx verdaccio`——运行私有注册表