DevSheets

npm

Projects, packages and scripts with npm 12.

01 Start a project

```
npm init                  # asks questions
npm init -y               # accept defaults
npm init --scope=@mycompany
npm create vite@latest my-app
npm init -w packages/api  # new workspace
```

npm init <name> (or npm create) runs npm exec create-<name>. npm 12 sets no default license: use npm config set init-license "MIT".

02 package.json

```
{
  "name": "my-project",
  "version": "1.0.0",
  "type": "module",
  "main": "index.js",
  "scripts": { "start": "node index.js" },
  "dependencies": { "express": "^4.18.0" },
  "devDependencies": { "jest": "^29.0.0" }
}
```

The name must be lowercase and URL-safe. "type": "module" makes .js files ES modules. ^4.18.0 accepts later 4.x versions; ~4.18.0 only 4.18.x.

03 Run tools with npx

```
npx eslint .              # project's eslint
npx prettier@3 --write .  # a chosen version
npx -p @angular/cli ng new app
npx -y npm-check-updates  # skip the prompt
```

npx (npm exec) runs the version installed in the project; otherwise it asks, then fetches into the npm cache. Add @latest for the newest; -p names the providing package.

04 Install packages

```
npm install           # all of package.json
npm i express         # add a dependency
npm i express@4.18.0  # a chosen version
npm i -D jest         # devDependencies
npm i -E express      # exact, no ^ range
npm i -g typescript   # global
```

npm i is short for npm install. Adding a package also updates package-lock.json. Keep tools in devDependencies and run them with npx: global installs can clash between projects.

05 Lockfile installs & sources

```
npm ci                  # clean install
npm install --omit=dev  # no devDependencies
npm i ../local-package  # symlinks a folder
npm i ./package.tgz
npm i github:user/repo --allow-git=root
npm install --legacy-peer-deps
```

npm ci needs package-lock.json, wipes node_modules and fails if the lock and package.json disagree. --omit=dev replaced --production. --legacy-peer-deps ignores peer conflicts. npm 12 blocks git installs unless allow-git is set.

06 Install scripts (npm 12)

```
npm install-scripts ls  # not yet reviewed
npm install-scripts approve sharp
npm install-scripts approve --all
npm install-scripts deny some-package
npm rebuild             # runs approved ones
```

npm 12 skips dependency install scripts until you approve them. Approvals live in allowScripts in package.json; global installs and npx take --allow-scripts=sharp.

07 Update packages

```
npm outdated              # Wanted vs Latest
npm update                # within ranges
npm update express
npm i express@latest      # past the range
npx npm-check-updates -u  # rewrites ranges
npm install
```

Wanted is the newest version your package.json range allows; Latest is the registry's latest tag. npm update leaves the ranges in package.json alone unless you add --save.

08 Remove & clean up

```
npm uninstall express     # or: remove, rm
npm uninstall -g typescript
npm uninstall express --no-save
npm prune                 # drop extraneous
npm dedupe                # merge duplicates
npm cache verify
```

prune removes installed packages that nothing depends on. For a clean reinstall run npm ci. npm cache clean --force is rarely needed: the cache checks its own integrity.

09 Run scripts

```
npm run build
npm run                   # list the scripts
npm start                 # no "run" needed
npm test                  # or stop, restart
npm run build -- --watch  # for the script
npm run lint && npm run build
```

prebuild and postbuild run automatically around npm run build; the same works for any script. npx concurrently "npm:dev:*" runs scripts in parallel.

DevSheets

npm

Config, publishing, audits and workspaces.

10 Config

```
npm config list  # -l adds the defaults
npm config get registry
npm config set save-exact true
npm config delete save-exact
npm config edit  # opens the user .npmrc
npm install --registry http://localhost:4873
```

Precedence: flag, npm_config_* environment variable, project .npmrc, user ~/.npmrc, global. config set writes the user file; --location=project targets the project.

11 Project .npmrc

```
# .npmrc in the project root
save-exact=true
engine-strict=true
fund=false
@mycompany:registry=https://npm.company.com/
```

Never commit auth tokens. npm 12 rejects unknown and abbreviated command-line flags such as --prod, and warns about unknown .npmrc keys.

12 Version bumps

```
npm version patch  # 1.0.0 -> 1.0.1
npm version minor  # 1.0.0 -> 1.1.0
npm version major  # 1.0.0 -> 2.0.0
npm version prerelease --preid=beta
npm version patch -m "Release v%s"
npm version patch --no-git-tag-version
```

Major is breaking, minor adds features, patch fixes. In a clean git repo it also commits and tags. From 1.0.0, prerelease --preid=beta gives 1.0.1-beta.0. A preversion script can run npm test first.

13 Publish

```
npm login                # web sign-in
npm publish --dry-run    # preview the files
npm publish              # tagged "latest"
npm publish --tag beta   # under another tag
```

A public scoped package needs --access public on its first publish. A version can never be reused. Unpublish is limited after 72 hours: prefer npm deprecate.

14 Fields for published packages

```
{
  "exports": {
    ".": {
      "import": "./dist/index.mjs",
      "require": "./dist/index.js"
    }
  },
  "files": ["dist", "README.md"],
  "bin": { "my-cli": "./cli.js" },
  "engines": { "node": ">=18.0.0" }
}
```

files is an allowlist for the tarball. engines only warns unless the user sets engine-strict.

15 Security audit

```
npm audit              # report known issues
npm audit fix          # compatible updates
npm audit fix --force  # allows major bumps
npm audit --omit=dev --audit-level=moderate
npm audit signatures   # verify packages
```

audit exits non-zero on any finding; --audit-level sets the minimum severity (low to critical).

16 Inspect packages

```
npm view express           # registry info
npm view express version   # latest version
npm view express versions  # every version
npm ls                     # top level only
npm explain express        # why it is here
npm fund                   # funding links
```

npm ls --all shows the whole tree; npm ls -g the global packages. npm why is an alias for explain.

17 Workspaces

```
{ "workspaces": ["packages/*", "apps/*"] }
```

```
npm install express -w packages/api
npm install @myapp/core -w apps/web
npm run build -w packages/core
npm run build --workspaces --if-present
npm publish --workspaces
```

Workspaces are symlinked into the root node_modules. Depend on a sibling by name with a normal range: npm has no workspace: protocol.

18 Link local packages

```
cd my-package && npm link  # global symlink
cd ../my-app
npm link my-package        # use it here
npm unlink my-package      # remove it again
npm link ../my-package     # both steps
npm ls -g --link           # global links
```

Links are not saved to package.json. npm rm -g my-package drops the global link. A "file:../my-package" dependency symlinks too; npm has no link: protocol.