# Package managers (npm, yarn)

## 📦 JavaScript Package Managers: npm & yarn – Detailed Explanation

Package managers help JavaScript developers **install, share, and manage** libraries (a.k.a. "packages" or "modules") in their projects. The two most popular ones are:

- **npm (Node Package Manager)**
- **Yarn (Yet Another Resource Negotiator)**

---

## 🔷 1. What is a Package Manager?

A package manager:

- Downloads packages from a **registry** (like npm registry)
- Resolves **dependencies**
- Installs them in your project (usually in `node_modules`)
- Manages **versioning** (semver)
- Runs scripts like `build`, `start`, `test`, etc.

---

## 🔷 2. What is npm?

### 📍 Overview:

- Default package manager for **Node.js**
- Ships **automatically** with Node.js
- Uses `package.json` to manage project metadata and dependencies

### 👉 Basic npm Commands:

```
npm init          # Creates package.json interactively
npm init -y        # Creates package.json with default values
npm install lodash   # Installs lodash package locally
npm install -g nodemon # Installs nodemon globally
npm uninstall lodash # Removes lodash
```

### 📁 Key Files and Folders:

- `node_modules/` → Where installed packages live
- `package.json` → Lists project dependencies
- `package-lock.json` → Exact versions installed

### ✅ Pros:

- Huge ecosystem
- Integrated with Node.js
- Active community

---

## 🔷 3. What is Yarn?

### 📍 Overview:

- Developed by **Facebook** as a faster, more secure alternative to npm
- Uses `yarn.lock` to lock versions (similar to `package-lock.json`)

### 👉 Basic Yarn Commands:

```
yarn init           # Initialize package.json
yarn add lodash      # Install lodash
yarn remove lodash   # Uninstall lodash
yarn global add nodemon # Install package globally
```

### ✅ Advantages over npm (historically):

- Faster due to parallel installation
- Better caching

- More deterministic installs via `yarn.lock`
- Workspaces for monorepos (multi-package projects)

> 🔄 Note: Since npm v7+, most of these gaps have closed — npm added support for workspaces, better performance, etc.

---

## 🔷 4. Comparison Table

| Feature | npm | Yarn |
|---|---|---|
| Developer | npm, Inc. (Node.js) | Facebook |
| Default with Node.js | ✅ | ❌ |
| Lock File | `package-lock.json` | `yarn.lock` |
| Speed (historically) | Slower | Faster (parallel downloads) |
| Caching | Less aggressive | Aggressive |
| Workspaces | ✅ (since v7) | ✅ |
| Plug'n'Play (PnP) | ❌ | ✅ |
| Offline installs | Limited | Better |

---

## 🔷 5. Scripts in package.json

Both npm and yarn can run scripts like:

```
{
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "test": "jest"
  }
}
```

### Run with:

```
npm run dev
# or
yarn dev
```

## 🔷 6. Versioning (SemVer)

Dependencies use **Semantic Versioning** (SemVer):

- `^1.2.3` → Compatible with `1.x.x` (non-breaking updates)
- `~1.2.3` → Only patch updates (e.g., `1.2.x`)
- `1.2.3` → Exact version

## 🔷 7. Global vs Local Installation

- **Local install** (default): in `node_modules/`, listed in `package.json`
- **Global install**: available system-wide (`g` flag)

## 🔷 8. Monorepo Support

Both npm (v7+) and Yarn support **monorepos** via **workspaces**:

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

Useful for managing multiple packages in one repo (e.g., frontend + backend).

## ✅ Conclusion

| Use npm if... | Use Yarn if... |
|---|---|
| You want simplicity and default | You want speed, PnP, or monorepos |
| You're already using Node.js tools | You're working in large teams |
| You want community-standard tools | You need offline or deterministic installs |

📦 **npm is still the most widely used**, but **Yarn** offers advanced features and performance benefits, especially for large-scale apps.

---

Would you like a visual diagram comparing npm and Yarn workflows?