Bun JS runtime and toolkit

START HERE

`bun init`
Scaffold a project

`bun <file>`
Run a TS or JS file

`bun install`
Install dependencies

`bun test`
Run the tests

`bun add <pkg>`
Add a dependency

`bun build <entry> --outdir <dir>`
Bundle for output

`bun run <script>`
Run a package script

`bunx <pkg>`
Run a package binary

package.json declared ranges → `bun install` → **bun.lock** pinned versions → `fetch + link` → **node_modules** global cache → `bun run / test` → **Your code** TS, JSX, JS

CONCEPTS

Bun is one binary: runtime, package manager, test runner and bundler. It runs TypeScript and JSX directly, no build step.

Put Bun flags right after bun: 'bun --watch run dev'. After the script name they go to the script.

Dependency lifecycle scripts (postinstall) are blocked unless the package is in trustedDependencies.

INSTALL AND ADD

`bun install`
Install all dependencies

`bun add <pkg>`
Add a dependency

`bun add <pkg>@<version>`
Add a specific version or tag

`bun add -d <pkg>`
Add a dev dependency

`bun add --exact <pkg>`
Pin the exact version

`bun add --optional <pkg>`
Add an optional dependency

`bun add --peer <pkg>`
Add a peer dependency

`bun add github:<user>/<repo>`
Add from a GitHub repo

`bun add -g <pkg>`
Install a CLI tool globally

`bun install --frozen-lockfile`
Fail if lockfile is out of date

`bun ci`
Same as install --frozen-lockfile

`bun install --production`
Skip devDependencies
Also implies --frozen-lockfile.

`bun install --filter <pattern>`
Install only for matching workspaces

`bun install --linker isolated`
Use isolated node_modules layout

`bun install --dry-run`
Resolve without installing

INSTALL AND ADD (cont.)

`bun install --ignore-scripts`
Skip all lifecycle scripts

REMOVE AND UPDATE

`bun remove <pkg>`
Remove a dependency

`bun update`
Update within package.json ranges

`bun update <pkg>`
Update one package

`bun update --latest`
Ignore ranges, take latest

`bun update -i`
Pick updates interactively

`bun update -r`
Update every workspace

`bun outdated`
Show packages with newer versions

`bun audit`
Check for known vulnerabilities

`bun audit fix`
Upgrade vulnerable packages

RUN AND WATCH

`bun <file>`
Run a TS, JS or JSX file

`bun run <script>`
Run a package.json script

`bun run`
List available scripts

`bun --watch <file>`
Restart on file change

`bun --hot <file>`
Reload in place, keep state

`bun --env-file=<file> <file>`
Load a specific .env file

`bun run --bun <bin>`
Run a Node-shebang binary with Bun

`bunx <pkg>`
Run a package binary without installing

`bunx <pkg>@<version>`
Run a specific version

`bunx -p <pkg> <bin>`
Binary name differs from package

RUN AND WATCH (cont.)

`bunx --bun <pkg>`
Run with Bun instead of Node

TEST

`bun test`
Run all tests

`bun test <path>`
Run files matching a path filter

`bun test -t '<regex>'`
Run tests by name pattern

`bun test --watch`
Re-run on file change

`bun test --coverage`
Print a coverage report

`bun test --bail`
Stop at the first failure

`bun test --timeout <ms>`
Set per-test timeout
Default is 5000 ms.

`bun test -u`
Update snapshots

BUILD AND SCAFFOLD

`bun init -y`
Scaffold with default answers

`bun build <entry> --outdir <dir>`
Bundle to a directory

`bun build <entry> --outfile <file>`
Bundle to one file

`bun build <entry> --compile --outfile <name>`
Make a standalone executable

`bun build <entry> --target <env>`
Target browser, bun or node

`bun build <entry> --minify`
Minify the output

`bun build <entry> --sourcemap=linked`
Emit source maps

WORKSPACES AND PACKAGES

`bun run --filter '<pattern>' <script>`
Run a script in matching workspaces

`bun run --workspaces <script>`
Run a script in all workspaces

WORKSPACES AND PACKAGES (cont.)

`bun run --parallel <script> <script>`
Run scripts concurrently

`bun add <pkg> --filter <workspace>`
Add a dependency to one workspace

`bun add <pkg> --catalog`
Record the version in the root catalog

`bun link`
Register this package for linking

`bun link <pkg>`
Link a registered package here

`bun pm ls`
List top-level installed packages

`bun why <pkg>`
Explain why a package is installed

`bun pm untrusted`
List blocked lifecycle scripts

`bun pm trust <pkg>`
Run blocked scripts and trust a package

Workspaces in package.json

```
"workspaces": {
  "packages": ["packages/*"],
  "catalog": { "react":
"^19.0.0" }
}
// in a package: "react":
"catalog:"
```

Allow a dependency script

```
"trustedDependencies":
["esbuild"]
# then reinstall, or: bun pm
trust esbuild
```

RECIPES

Start a TypeScript project

```
bun init -y
bun add zod
bun add -d @types/node
bun --watch index.ts
```

CI install and test

```
bun ci
bun test --bail
bun run build
```

Ship a single binary

```
bun build ./cli.ts --compile --
outfile mycli
./mycli --help
```

Monorepo: one workspace

```
bun install
bun add zod --filter api
bun run --filter api dev
```