Deno Secure TS runtime

START HERE

`deno init`
New project

`deno fmt && deno lint`
Format and lint

`deno run main.ts`
Run a script

`deno compile main.ts`
Build one executable

`deno run -R -N main.ts`
Run with read + net

`deno task <name>`
Run a config task

`deno add jsr:@std/assert`
Add a dependency

`deno test`
Run tests

Specifier jsr: npm: https: → `resolve` → **Cache** deno.lock pins it → `check` → **Type-check** TypeScript built in → `permissions` → **Run** sandboxed by flags

CONCEPTS

Scripts run in a sandbox: no disk, network or env access unless you grant it with --allow-* flags (or answer the prompt).

RUN AND PERMISSIONS

`deno run <file>`
Run a script or URL

`deno run -A <file>`
Allow everything (trusted code only)
Disables the sandbox; fine for local scripts you wrote, risky for anything fetched.

`--allow-read[=<path>] / -R`
Allow file reads, optionally scoped

`--allow-write[=<path>] / -W`
Allow file writes

`--allow-net[=<host>] / -N`
Allow network access

`--allow-env[=<VAR>] / -E`
Allow reading env vars

`--deny-read=<path>`
Deny a path even if allowed more broadly

RUN AND PERMISSIONS (cont.)

`--no-prompt`
Fail instead of asking for permission

`deno run --watch <file>`
Restart on file changes

`--env-file[=<path>]`
Load env vars from a file

`deno eval "<code>"`
Evaluate a snippet

`deno repl`
Interactive prompt

`deno serve <file>`
Run a default-export fetch handler as a server

`--inspect-brk`
Debugger: pause on first line

TASKS AND CONFIG

`deno task`
List tasks in deno.json

`deno task <name>`
Run a task

`deno task "<prefix>:*"`
Run all matching tasks in parallel
Quote the pattern so the shell does not expand it.

TASKS AND CONFIG (cont.)

`deno task -r <name>`
Run a task in every workspace member

`deno init <dir>`
New project with main.ts and a test

deno.json task forms

Form	Meaning
`"build": "deno run ..."`	Plain command string
`"command"`	Command in object form
`"description"`	Shown in task list
`"dependencies"`	Tasks run first, in parallel

DEPENDENCIES

`deno add <pkg>`
Add dependencies to the config

`deno add jsr:@std/<name>`
Add a JSR package

`deno add npm:<pkg>`
Add an npm package

DEPENDENCIES (cont.)

`deno install`
Install all deps from deno.json/ package.json

`deno remove <pkg>`
Remove a dependency

`deno outdated`
Show newer dependency versions

`deno outdated --update`
Update within semver-compatible range

`deno info [<file>]`
Show a module's dependency tree

`deno audit`
Scan dependencies for known vulnerabilities

TEST AND CHECK

`deno test`
Run all tests

`deno test <file>`
Run one test file

`deno test --filter "<name>"`
Run tests matching a name
Wrap a regex in slashes, e.g. "/^db/".

`deno test --coverage`
Collect coverage data

TEST AND CHECK (cont.)

`deno test --parallel`
Run test files on several threads

`deno check <file>`
Type-check without running

> A test is Deno.test("name", () => {...}) in a file named *_test.ts or *.test.ts.

FORMAT AND LINT

`deno fmt`
Format all files

`deno fmt --check`
Fail if anything is unformatted

`deno lint`
Lint all files

`deno lint --fix`
Auto-fix what rules allow

BUILD AND PUBLISH

`deno compile <file>`
Bundle into a standalone executable

`deno compile -o <name> <file>`
Name the output

BUILD AND PUBLISH (cont.)

`deno compile --target <triple>`
Cross-compile for another platform

`deno upgrade`
Update Deno itself

RECIPES

Script with scoped permissions

```
# read one folder, call one API, load .env
deno run --allow-read=./data --allow-net=api.example.com \
  --env-file main.ts
```

Start a TypeScript project

```
deno init my-app && cd my-app
deno add jsr:@std/assert
deno task dev
deno test
```

CI gate

```
deno fmt --check
deno lint
deno check .
deno test --coverage --fail-fast
deno audit
```

Ship a CLI as one binary

```
deno compile --allow-read --allow-net -o mytool main.ts
deno compile --target aarch64-apple-darwin -o mytool-mac main.ts
```