DevSheets

# Bun

Runtime, package manager and server in one binary.



## 01 Install & create

```
curl -fsSL https://bun.com/install | bash
brew install oven-sh/bun/bun  # or Homebrew
bun init                      # new project
bun create hono my-api        # template
bun upgrade                   # update Bun
```

Windows: powershell -c "irm bun.sh/install.ps1|iex". bun init writes package.json, tsconfig.json and index.ts. Bun runs .ts, .tsx and .jsx with no transpiler setup.

## 02 Run files & scripts

```
bun index.ts            # run a file
bun run dev             # package.json script
bun --watch index.ts    # restart on change
bun --hot server.ts     # reload, keep state
bun --filter '*' dev    # every workspace
bunx cowsay "Hello!"    # like npx
bunx --bun vite         # Bun, not Node
```

Also: --env-file=.env.local, --smol (less memory) and --inspect (debugger).

## 03 Package manager

```
bun install             # install all deps
bun add zod             # dependency
bun add -d typescript   # devDependency
bun add zod@3.22.0      # exact version
bun remove zod
bun update              # update all
bun outdated            # what is stale
bun why zod             # who depends on it
```

Commit bun.lock, the text lockfile (default since 1.2). In CI: bun install --frozen-lockfile.

## 04 Environment variables

```
const db = Bun.env.DATABASE_URL;
const key = process.env.API_KEY;
const port = Bun.env.PORT ?? "3000";
import.meta.dir;   // this file's folder
import.meta.path;  // this file's path
```

Auto-loaded, no dotenv: .env, .env.{NODE_ENV}, then .env.local (later wins). Bun.env aliases process.env.

## 05 Bun.serve() routes

```
const server = Bun.serve({
  routes: {
    "/": new Response("Hello!"),
    "/users/:id": (req) =>
      Response.json({ id: req.params.id }),
    "/logo.png": Bun.file("./logo.png"),
  },
});
```

Port: $PORT or 3000. "/api/*" is a wildcard. fetch(req) gets unmatched paths; error(err) gets throws.

## 06 Methods & HTML imports

```
import home from "./index.html";
Bun.serve({
  development: true,  // HMR + overlay
  routes: {
    "/": home,        // JS, CSS bundled
    "/api/posts": { GET: list, POST: add },
  },
});
```

The routes option needs Bun 1.2.3+. server.reload() swaps routes, fetch and error without a restart; the port stays.

## 07 WebSockets

```
Bun.serve({
  fetch(req, server) {
    if (server.upgrade(req)) return;
    return new Response("Plain HTTP");
  },
  websocket: {
    open(ws) { ws.subscribe("chat"); },
    message(ws, msg) {
      ws.publish("chat", msg);
    },
  },
});
```

publish() reaches every subscriber but the sender. upgrade(req, { data }) sets ws.data.

## 08 Read & write files

```
const file = Bun.file("a.json");  // lazy
await file.exists();              // boolean
const data = await file.json();   // parsed
await Bun.write("out.txt", "Hello");
await Bun.write("b.json", file);  // copy
```

Nothing is read until .text(), .json() or .stream(). To append, use appendFile from node:fs/promises.

## 09 Hashing & passwords

```
const hash = await Bun.password.hash(pw);
await Bun.password.verify(pw, hash);
Bun.hash("hello");      // fast, not crypto
const sha = new Bun.CryptoHasher("sha256");
sha.update("hello world").digest("hex");
```

Passwords default to argon2id; bcrypt also works.

# Bun

Data, tests, builds and processes.



## 10 bun:sqlite

```
import { Database } from "bun:sqlite";
const db = new Database("app.db");
db.exec("CREATE TABLE users (name TEXT)");
const add = db.prepare(
  "INSERT INTO users (name) VALUES (?)");
add.run("Maya");
db.query("SELECT * FROM users").all();
```

Bind with ? or $name, never string building. get() returns one row; db.transaction(fn) batches writes atomically.

## 11 Tests

```
import { expect, test } from "bun:test";
test("math", () => expect(2 + 2).toBe(4));
test("api", async () => {
  const res = await fetch(url);
  expect(res.ok).toBeTrue();
}, 10_000);  // timeout in ms
```

Run bun test (--watch reruns on save). describe, beforeAll, afterEach, test.skip and test.todo work as in Jest.

## 12 Mocks & snapshots

```
import { expect, mock, spyOn,
  setSystemTime } from "bun:test";
const fn = mock(() => 42);
fn();
expect(fn).toHaveBeenCalledTimes(1);
const log = spyOn(console, "log");
mock.module("./db", () => ({ query: fn }));
setSystemTime(new Date("2026-01-01"));
```

toMatchSnapshot() saves on the first run; bun test --update-snapshots rewrites. mockRestore() undoes a spy.

## 13 bun build

```
bun build ./index.ts --outdir dist
bun build ./index.ts --outdir dist --minify
bun build ./index.ts --outdir dist \
  --target bun --sourcemap=external
bun build ./index.ts --compile --outfile app
```

The default target is browser; use bun or node for servers. --splitting shares chunks between entry points. --compile makes one executable that runs without Bun.

## 14 Bun.build() API

```
const result = await Bun.build({
  entrypoints: ["./src/index.ts"],
  outdir: "./dist",
  minify: true,
  env: "inline",  // bake in process.env
});
result.outputs.map((file) => file.path);
```

A failed build rejects with an AggregateError, so use try/catch, or pass throw: false and check result.success.

## 15 Bun Shell ($)

```
import { $ } from "bun";
await $`echo "Hello"`;           // prints
const out = await $`ls`.text();  // capture
const dir = "my files";
await $`ls ${dir}`;              // escaped
await $`npm install`.quiet();    // silent
const r = await $`grep x a.txt`.nothrow();
console.log(r.exitCode);         // 0 or 1
```

Non-zero exits throw unless .nothrow(). .json() parses output; .lines() is an async iterator.

## 16 Bun.spawn()

```
const proc = Bun.spawn(["ls", "-la"], {
  cwd: "/tmp",
  env: { ...process.env, DEBUG: "1" },
});
const out = await proc.stdout.text();
const code = await proc.exited;
const r = Bun.spawnSync(["git", "status"]);
```

Args are an array, so no shell. stdout pipes by default.

## 17 Workers

```
// main.ts
const w = new Worker("./worker.ts");
w.postMessage(21);
w.onmessage = (e) => console.log(e.data);
// worker.ts
declare var self: Worker;
self.onmessage = (e: MessageEvent) =>
  self.postMessage(e.data * 2);
```

Workers run TypeScript with no build step, on their own thread. Bun still marks the Worker API experimental.

## 18 bunfig.toml

```
preload = ["./setup.ts"]  # runs first
[install]
exact = true              # no ^ or ~
[test]
coverage = true
coverageThreshold = 0.8   # else exit 1
[run]
bun = true                # node runs as bun
```

~/.bunfig.toml holds global settings.