readme.md for @git.zone/tsrun

npm version License: MIT TypeScript Node.js

> Run TypeScript files instantly — no build step, no config, no friction ⚡

Execute TypeScript programs on-the-fly with zero configuration. Whether you're writing quick scripts, prototyping ideas, or orchestrating complex multi-project workflows, tsrun gets out of your way and lets you focus on code.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Table of Contents

What is tsrun?

tsrun is a lightweight TypeScript execution tool that lets you run `.ts` files directly — just like running JavaScript with `node`, but for TypeScript. Under the hood, it uses tsx for lightning-fast transpilation, so there's no compilation step, no tsconfig fiddling, and no waiting around.

It also doubles as a **programmatic library** with full process control — spawn TypeScript processes, capture their output, set timeouts, and cancel them with `AbortController`. Perfect for build scripts, task runners, and orchestration tools.

Installation

Global (recommended for CLI usage):

```
pnpm install -g @git.zone/tsrun
```

As a project dependency (for programmatic API):

```
pnpm install @git.zone/tsrun
```

□□ CLI Usage

Run any TypeScript file:

```
tsrun myScript.ts
```

Arguments pass through transparently, just like `node`:

```
tsrun deploy.ts production --verbose --dry-run
```

Your script sees them in `process.argv` as expected:

```
// deploy.ts
const env = process.argv[0];     // "production"
const verbose = process.argv.includes('--verbose');
const dryRun = process.argv.includes('--dry-run');

console.log(`Deploying to ${env}...`);
```

□□ Programmatic API

tsrun exports three functions tailored for different execution needs.

`runPath()` — Simple Execution

Runs a TypeScript file and waits for it to complete. The simplest way to execute scripts programmatically.

```
import { runPath } from '@git.zone/tsrun';

// Run a script (path relative to cwd)
await runPath('./scripts/build.ts');

// Resolve path relative to the calling file
await runPath('./build.ts', import.meta.url);

// Run in a different working directory (spawns a child process)
await runPath('./build.ts', import.meta.url, { cwd: '/path/to/project' });
```

How it works:

- Without `cwd` — executes **in-process** using tsx's ESM loader (fast, zero overhead)
- With `cwd` — spawns an **isolated child process** with the given working directory

`runCli()` — CLI Mode

Runs with `process.argv` integration, as if the script were invoked from the command line. This is what the `tsrun` CLI binary uses internally.

```
import { runCli } from '@git.zone/tsrun';

// Uses process.argv for argument passing
await runCli('./script.ts');

// With custom working directory
await runCli('./script.ts', { cwd: '/path/to/project' });
```

`spawnPath()` — Advanced Process Control

Returns immediately with a process handle, giving you full control over stdio, timeouts, and cancellation.

```
import { spawnPath } from '@git.zone/tsrun';

const proc = spawnPath('./task.ts', import.meta.url, {
  timeout: 30000,                    // Kill after 30s
  cwd: '/path/to/project',
  env: { NODE_ENV: 'production' },   // Merged with process.env
  args: ['--verbose'],               // Extra CLI args
  stdio: 'pipe',                     // Default: capture stdout/stderr
});

// Stream stdout
proc.stdout?.on('data', (chunk) => {
  console.log(chunk.toString());
});

// Wait for exit
const exitCode = await proc.exitCode;
```

AbortController support:

```
const controller = new AbortController();
const proc = spawnPath('./task.ts', import.meta.url, {
  signal: controller.signal,
});

// Cancel from outside
```

```
setTimeout(() => controller.abort(), 5000);

try {
  await proc.exitCode;
} catch (err) {
  console.log('Process was cancelled');
}
```

Graceful termination:

```
const proc = spawnPath('./server.ts', import.meta.url);

// Sends SIGTERM, waits 5s, then SIGKILL if still running
await proc.terminate();
```

API Quick Reference

Function	Execution	Returns	Best For
`runPath()`	In-process (or child with `cwd`)	`Promise<void>`	Simple script execution, sequential workflows
`runCli()`	In-process (or child with `cwd`)	`Promise<void>`	CLI-like invocation with argv integration
`spawnPath()`	Always child process	`ITsrunChildProcess`	Output capture, timeouts, cancellation, parallel tasks

Decision guide:

- □□ **Just run a script?** → `runPath()`
- □□ **Need argv pass-through?** → `runCli()`
- □□ **Need stdout/stderr, timeout, or cancel?** → `spawnPath()`
- ⚡ **Parallel execution across projects?** → `runPath()` with `cwd` or `spawnPath()`

`ITsrunChildProcess` Interface

The object returned by `spawnPath()`:

Property / Method	Type	Description
`childProcess`	`ChildProcess`	Direct access to Node's ChildProcess

Property / Method	Type	Description
`stdout`	`Readable | null`	Stdout stream (`null` if stdio is `'inherit'`)
`stderr`	`Readable | null`	Stderr stream (`null` if stdio is `'inherit'`)
`exitCode`	`Promise<number>`	Resolves with exit code when process ends
`kill(signal?)`	`(signal?: NodeJS.Signals) => boolean`	Send a signal to the process
`terminate()`	`() => Promise<void>`	Graceful shutdown: SIGTERM → 5s → SIGKILL

`ISpawnOptions`

Options for `spawnPath()`:

Option	Type	Default	Description
`cwd`	`string`	`process.cwd()`	Working directory for the child process
`env`	`Record<string, string>`	`{}`	Extra env vars (merged with `process.env`)
`args`	`string[]`	`[]`	Additional CLI arguments
`stdio`	`'pipe' | 'inherit'`	`'pipe'`	Stdio configuration
`timeout`	`number`	—	Auto-kill after N milliseconds
`signal`	`AbortSignal`	—	External cancellation support

Features

 Zero Configuration — Point and shoot. No tsconfig required, no build step, no setup.

⚡ **Lightning Fast** — Powered by tsx (esbuild under the hood) for near-instant TypeScript execution.

 Transparent Arguments — CLI args pass through seamlessly to your scripts via `process.argv`.

 Dual Interface — Use as a CLI tool or import as a library with full TypeScript types.

 Custom Working Directory — Run scripts in isolated child processes with different cwds.

Full Process Control — `spawnPath()` gives you streams, timeouts, cancellation, and graceful shutdown.

Signal Forwarding — SIGINT/SIGTERM/SIGHUP are properly forwarded to child processes.

Common Use Cases

Quick Scripts & Prototyping

```
# Write TypeScript, run it immediately
tsrun seed-database.ts
tsrun generate-report.ts --format csv
tsrun cleanup-temp-files.ts
```

Sequential Build Pipeline

```
import { runPath } from '@git.zone/tsrun';

const steps = ['./lint.ts', './test.ts', './build.ts', './deploy.ts'];

for (const step of steps) {
  console.log(`▶ Running ${step}...`);
  await runPath(step, import.meta.url);
  console.log(`✓ Done`);
}
```

Parallel Multi-Project Builds

```
import { runPath } from '@git.zone/tsrun';

await Promise.all([
  runPath('./build.ts', undefined, { cwd: '/workspace/frontend' }),
  runPath('./build.ts', undefined, { cwd: '/workspace/backend' }),
  runPath('./build.ts', undefined, { cwd: '/workspace/shared' }),
]);
```

Long-Running Tasks with Monitoring

```
import { spawnPath } from '@git.zone/tsrun';

const proc = spawnPath('./data-migration.ts', import.meta.url, {
  timeout: 300000,  // 5 minute max
  env: { LOG_LEVEL: 'verbose' },
});

let lines = 0;
proc.stdout?.on('data', (chunk) => {
  lines++;
  if (lines % 100 === 0) console.log(`Processed ${lines} lines...`);
});

try {
  await proc.exitCode;
  console.log('Migration completed!');
} catch (err) {
  console.error('Migration failed:', err.message);
  process.exit(1);
}
```

Troubleshooting

"Cannot find module" errors

Use `import.meta.url` for path resolution relative to the calling file:

```
// ❌ Relative to cwd — fragile
await runPath('./script.ts');

// ✅ Relative to current file — reliable
await runPath('./script.ts', import.meta.url);
```

Process hangs

When using `spawnPath()`, always await the `exitCode` promise:

```
const proc = spawnPath('./script.ts', import.meta.url);
await proc.exitCode; // Don't forget this!
```

Timeout only works with `spawnPath()`

`runPath()` executes in-process and doesn't support timeouts. Use `spawnPath()` instead:

```
const proc = spawnPath('./script.ts', import.meta.url, { timeout: 5000 });
await proc.exitCode;
```

`tsrun: command not found`

Install globally or use `npx`:

```
pnpm install -g @git.zone/tsrun
# or
npx @git.zone/tsrun myScript.ts
```

License and Legal Information

Trademarks

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #5
Created 2026-03-28 10:49:33 UTC by foss.global Team
Updated 2026-03-28 12:15:31 UTC by foss.global Team