Node.js CLI Project Structure

Command-line application with TypeScript, commander, and npm publishing setup.

#node #nodejs #cli #command-line #typescript

PNG PDF Copy Prompt

Project Directory

```
mycli/
  package.json  Package config …
  tsconfig.json
  LICENSE
  README.md
  .gitignore
  .npmignore  Exclude src fro…
  src/
    index.ts  CLI entry with …
    cli.ts  Commander progr…
    commands/  Subcommand hand…
      index.ts
      init.ts  mycli init
      run.ts  mycli run
    lib/  Core business l…
      index.ts
      config.ts  Config file han…
      runner.ts
    utils/
      index.ts
      logger.ts  Colored console…
      prompts.ts  Interactive pro…
  bin/  Compiled entry …
    mycli.js  Built output
  tests/
    cli.test.ts
    commands/
      init.test.ts
```

Why This Structure?

Node.js CLIs leverage the npm ecosystem and are easy to distribute via `npx`. This structure uses TypeScript for type safety and commander for argument parsing. The `bin/` field in package.json enables global installation.

Key Directories

`src/index.ts` - Entry point with #!/usr/bin/env node

`src/cli.ts` - Commander program and command registration

`src/commands/` - One file per subcommand

`bin/` - Compiled JS, referenced in package.json bin

Commander Setup

```
// src/cli.ts
import { Command } from 'commander';
import { initCommand } from './commands/init';
import { runCommand } from './commands/run';

const program = new Command()
  .name('mycli')
  .description('CLI description')
  .version('1.0.0');

program
  .command('init <name>')
  .description('Initialize a new project')
  .action(initCommand);

program
  .command('run')
  .option('-c, --config <path>', 'Config file')
  .action(runCommand);

export { program };
```

Getting Started

1. `npm init -y`
2. `npm add commander chalk`
3. `npm add -D typescript @types/node`
4. Add `"bin": { "mycli": "./bin/mycli.js" }` to package.json
5. `npm run build && npm link`

When To Use This

- Building npm-distributed CLI tools
- Want rich npm ecosystem (chalk, ora, inquirer)
- Need `npx mycli` execution
- Interactive prompts and spinners
- JavaScript/TypeScript team

Trade-offs

`Startup time` - Node.js has slower cold start than Go/Rust

`Distribution` - Requires Node.js installed (or bundle with pkg)

`Dependencies` - node_modules can grow large

Best Practices

- Add shebang `#!/usr/bin/env node` to entry
- Use `chalk` for colored output
- Use `ora` for spinners on long operations
- Handle SIGINT gracefully
- Provide `--json` flag for scriptable output

Naming Conventions

`Commands` - One file per command (init.ts, run.ts)

`Exports` - Named exports (initCommand, runCommand)

`Package name` - Lowercase with hyphens (my-cli)