# readme.md for @git.zone/cli

**The ultimate CLI toolbelt for modern TypeScript development workflows**

npm version License: MIT

# □□ What is gitzone?

gitzone is a powerful command-line interface that supercharges your development workflow with automated project management, intelligent code formatting, seamless version control, and development service orchestration. Whether you're bootstrapping a new TypeScript project, maintaining code quality, managing complex multi-repository setups, or spinning up local development databases, gitzone has got you covered.

# 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.

# □□♂ Quick Start

## Installation

```
# Install globally via pnpm (recommended)
pnpm add -g @git.zone/cli

# Or with npm
npm install -g @git.zone/cli
```

Once installed, you can use either `gitzone` or the shorter `gzone` command from anywhere in your terminal.

# Your First Commands

```
# Create a new TypeScript npm package
gitzone template npm

# Format your entire codebase (dry-run by default)
gitzone format

# Apply formatting changes
gitzone format --write

# Start local MongoDB and MinIO services
gitzone services start

# Create a semantic commit with AI-powered suggestions
gitzone commit
```

# □□ Core Features

## □□ Semantic Commits & Versioning

Create standardized commits with AI-powered suggestions that automatically handle versioning:

```
# Interactive commit with AI recommendations
gitzone commit

# Auto-accept AI recommendations (skipped for BREAKING CHANGEs)
gitzone commit -y

# Auto-accept, push, build, and release
gitzone commit -ypbr
```

**Flags:**

| Flag | Long Form | Description |
|---|---|---|
| `-y` | `--yes` | Auto-accept AI recommendations |
| `-p` | `--push` | Push to remote after commit |
| `-t` | `--test` | Run tests before committing |
| `-b` | `--build` | Build after commit, verify clean tree |
| `-r` | `--release` | Publish to configured npm registries |
|  | `--format` | Run format before committing |

**Workflow steps:**

1. □□ **AI-powered analysis** — analyzes your changes and suggests commit type, scope, and message
2. □□ Interactive commit message builder (type: `fix` / `feat` / `BREAKING CHANGE`, scope, description)
3. □□ Automatic changelog generation
4. □□ Automatic version bumping (major/minor/patch) with git tag creation
5. □□ Optional build & verification
6. □□ Optional push to origin
7. □□ Optional publish to npm registries

Supports both npm (`package.json`) and Deno (`deno.json`) projects, including dual-type projects.

# □□ Intelligent Code Formatting

Automatically format and standardize your entire codebase. **Dry-run by default** — nothing changes until you explicitly use `--write`:

```
# Preview what would change (default behavior)
gitzone format

# Apply changes
gitzone format --write

# Auto-approve without prompts
gitzone format --yes --write

# Show detailed diffs
gitzone format --diff

# Enable verbose logging
```

```
gitzone format --verbose
```

**Flags:**

| Flag | Description |
| --- | --- |
| `--write` / `-w` | Apply changes (default is dry-run) |
| `--yes` | Auto-approve without interactive confirmation |
| `--plan-only` | Only show what would be done |
| `--save-plan <file>` | Save the format plan to a file |
| `--from-plan <file>` | Load and execute a saved plan |
| `--detailed` | Show detailed stats and save report |
| `--parallel` / `--no-parallel` | Toggle parallel execution |
| `--verbose` | Enable verbose logging |
| `--diff` | Show file diffs |

**Formatters (executed in order):**

1. □□ **Cleanup** — removes obsolete files (yarn.lock, package-lock.json, tslint.json, etc.)
2. ⚙ **Npmextra** — formats and standardizes `npmextra.json`
3. □□ **License** — ensures proper licensing and checks dependency licenses
4. □□ **Package.json** — standardizes package configuration
5. □□ **Templates** — applies project template updates
6. □□ **Gitignore** — updates repository ignore rules
7. □□ **Tsconfig** — optimizes TypeScript configuration
8. □ **Prettier** — applies code formatting
9. □□ **Readme** — ensures readme files exist
10. □□ **Copy** — copies configured files

# □□ Development Services Management

Effortlessly manage local development services (MongoDB, MinIO S3, Elasticsearch) with Docker:

```
gitzone services [command]
```

**Commands:**

| Command | Description |
| --- | --- |
| `start [service]` | Start services (`mongo` \| `s3` \| `elasticsearch` \| `all`) |
| `stop [service]` | Stop services |

| Command | Description |
| --- | --- |
| `restart [service]` | Restart services |
| `status` | Show current service status |
| `config` | Display configuration details |
| `compass` | Get MongoDB Compass connection string with network IP |
| `logs [service] [lines]` | View service logs (default: 20 lines) |
| `reconfigure` | Reassign ports and restart all services |
| `remove` | Remove containers (preserves data) |
| `clean` | Remove containers AND data (⚠ destructive) |

**Service aliases:**

- `mongo` / `mongodb` — MongoDB
- `minio` / `s3` — MinIO (S3-compatible storage)
- `elasticsearch` / `es` — Elasticsearch
- `all` — All services (default)

**Key features:**

- □□ **Smart port assignment** — automatically assigns random ports (20000–30000) to avoid conflicts
- □□ **Project isolation** — each project gets its own containers with unique names
- □□ **Data persistence** — data stored in `.nogit/` survives container restarts
- □□ **MongoDB Compass support** — instantly get connection strings for GUI access
- □□ **Network IP detection** — detects your local network IP for remote connections
- ⚙ **Auto-configuration** — creates `.nogit/env.json` with smart defaults

**Global operations (`-g` flag):**

```
# List all registered projects
gitzone services list -g

# Show status across all projects
gitzone services status -g

# Stop all containers across all projects
gitzone services stop -g

# Remove stale registry entries
gitzone services cleanup -g
```

**Example workflow:**

```
# Start all services for your project
gitzone services start

# Check what's running
gitzone services status

# Get MongoDB Compass connection string
gitzone services compass
# Output: mongodb://defaultadmin:defaultpass@192.168.1.100:27018/myproject?authSource=admin

# View MongoDB logs
gitzone services logs mongo 50

# Stop services when done
gitzone services stop
```

# ⚙ Release & Commit Configuration

Manage release registries and commit settings:

```
gitzone config [subcommand]
```

| Command | Description |
|---|---|
| `show` | Display current release config (registries, access level) |
| `add [url]` | Add a registry URL (default: `https://registry.npmjs.org`) |
| `remove [url]` | Remove a registry URL (interactive selection if no URL) |
| `clear` | Clear all registries (with confirmation) |
| `access [public\|private]` | Set npm access level for publishing |
| `commit alwaysTest [true\|false]` | Always run tests before commit |
| `commit alwaysBuild [true\|false]` | Always build after commit |
| `services` | Configure which services are enabled |

Configuration is stored in `npmextra.json` under the `@git.zone/cli` key.

# □□ Project Templates

Instantly scaffold production-ready projects with best practices built-in:

```
gitzone template [template-name]
```

**Interactive templates:**

- `npm` — TypeScript npm package with testing, CI/CD, and full tooling
- `service` — Microservice architecture with Docker support
- `website` — Modern web application with LitElement and service workers
- `wcc` — Web Component Collection for reusable UI components

Each template comes pre-configured with:

- □ TypeScript with modern configurations
- □ Automated testing setup with `@git.zone/tstest`
- □ CI/CD pipelines (GitLab/GitHub)
- □ Code formatting and linting
- □ Documentation structure

# □□ Meta Repository Management

Manage multiple related repositories as a cohesive unit:

```
# Initialize a meta repository
gitzone meta init

# Add a sub-project
gitzone meta add [name] [git-url]

# Update all sub-projects (clone missing, clean superfluous)
gitzone meta update

# Remove a sub-project
gitzone meta remove [name]
```

# □□ Docker Management

Streamline your Docker workflow:

```
# Clean up all Docker resources (containers, images, volumes, networks)
gitzone docker prune
```

# □□ Quick CI/CD Access

Jump directly to your CI/CD configurations:

```
# Open CI/CD settings
gitzone open ci

# Open pipelines view
gitzone open pipelines
```

Works with GitLab repositories to provide instant access to your deployment configurations.

# □□ Package Deprecation

Smoothly transition users from old to new packages:

```
gitzone deprecate
```

Interactive wizard that prompts for registry URLs, old package name, and new package name — then runs `npm deprecate` across all specified registries.

# □□ Project Initialization

Prepare existing projects for development:

```
gitzone start
```

Automatically checks out master, pulls latest changes, and installs dependencies.

# □□ Helper Utilities

```
# Generate a unique short ID
gitzone helpers shortid
```

# ⚙️ Configuration

## npmextra.json

Customize gitzone behavior through `npmextra.json`:

```
{
  "@git.zone/cli": {
    "projectType": "npm",
    "release": {
      "registries": [
        "https://registry.npmjs.org"
      ],
      "accessLevel": "public"
    },
    "commit": {
      "alwaysTest": false,
      "alwaysBuild": false
    }
  },
  "gitzone": {
    "format": {
      "interactive": true,
      "parallel": true,
      "showStats": true,
      "cache": {
        "enabled": true,
        "clean": true
      },
      "modules": {
        "skip": ["prettier"],
        "only": [],
        "order": []
      },
      "licenses": {
        "allowed": ["MIT", "Apache-2.0"],
        "exceptions": {
```

```
          "some-package": "GPL-3.0"
        }
      }
    }
  }
}
```

# Environment Variables

- `CI` — Detect CI environment for automated workflows
- `DEBUG` — Enable debug output
- `GITZONE_FORMAT_PARALLEL` — Control parallel formatting

# □□ Common Workflows

## Full-Stack Development Cycle

```
# 1. Start fresh
gitzone start

# 2. Spin up databases and services
gitzone services start

# 3. Make changes
# ... your development work ...

# 4. Check service logs if needed
gitzone services logs mongo

# 5. Preview format changes, then apply
gitzone format
gitzone format --write

# 6. Commit with semantic versioning
gitzone commit
```

```
# 7. Stop services when done
gitzone services stop
```

# Automated CI/CD Commit

```
# Auto-accept, test, build, push, and release in one command
gitzone commit -ytbpr
```

# Multi-Repository Management

```
# 1. Set up meta repository
gitzone meta init

# 2. Add all related projects
gitzone meta add frontend https://github.com/org/frontend.git
gitzone meta add backend https://github.com/org/backend.git
gitzone meta add shared https://github.com/org/shared.git

# 3. Synchronize updates
gitzone meta update
```

# Safe Formatting with Plan Review

```
# 1. Preview changes (default)
gitzone format

# 2. Save plan for review
gitzone format --save-plan format-changes.json

# 3. Apply from saved plan
gitzone format --from-plan format-changes.json --write
```

# Database-Driven Development

```
# 1. Start MongoDB, MinIO, and Elasticsearch
gitzone services start

# 2. Get connection details
gitzone services config

# 3. Connect with MongoDB Compass
gitzone services compass

# 4. Monitor services
gitzone services status

# 5. Clean everything when done
gitzone services clean  # ⚠ Warning: deletes data
```

# Integrations

## CI/CD Platforms

- **GitLab CI** — full pipeline support with templates
- **GitHub Actions** — automated workflows
- **Docker** — container-based deployments

## Development Tools

- **TypeScript** — first-class support
- **Prettier** — code formatting
- **pnpm** — package management
- **MongoDB** — local database service
- **MinIO** — S3-compatible object storage
- **Elasticsearch** — search and analytics
- **MongoDB Compass** — database GUI integration

## Version Control

- **Git** — deep integration
- **Semantic Versioning** — automatic version bumping

- **Conventional Commits** — standardized commit messages
- **AI-Powered Analysis** — intelligent commit suggestions via `@git.zone/tsdoc`

# □□ Pro Tips

1. **Use aliases**: Add `alias gz='gitzone'` to your shell profile
2. **Combine flags**: `gitzone commit -ypbr` for the full auto workflow
3. **Leverage templates**: Start projects right with proven structures
4. **Enable caching**: Dramatically speeds up formatting operations
5. **Save format plans**: Review changes before applying
6. **Port management**: Let services auto-assign ports to avoid conflicts
7. **Use MongoDB Compass**: `gitzone services compass` for visual DB management
8. **Global service management**: `gitzone services status -g` to see all projects' services at once

# □□ Troubleshooting

## Format Command Shows "Cancelled"

- Check your `npmextra.json` configuration
- Try with `--yes --write` flags
- Use `--verbose` for detailed output

## Docker Commands Fail

Ensure Docker daemon is running:

```
docker info
```

## Services Won't Start

```
# Services auto-assign ports, but you can check the config
cat .nogit/env.json

# Verify Docker is running
docker ps
```

```
# Reassign ports if there are conflicts
gitzone services reconfigure
```

# Template Creation Issues

Verify pnpm/npm is properly configured:

```
npm config get registry
```

# MongoDB Connection Issues

- Ensure services are running: `gitzone services status`
- Check firewall settings for the assigned ports
- Use `gitzone services compass` for the correct connection string

# □□ Performance

gitzone is optimized for speed:

- ⚡ **Parallel processing** for format operations
- □□ **Smart caching** to avoid redundant work
- □□ **Incremental updates** for meta repositories
- □□ **Isolated services** prevent resource conflicts
- □□ **Auto port assignment** eliminates manual configuration

# License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

**Please note:** The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

# Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

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