CLI Reference

All `ori` commands

Overview

The `ori` CLI is your main tool for working with Origami Engine. After installation (`npm run begin`), the `ori` command is available globally.

Commands

`ori create`

Creates a new game project from a template.

Syntax: `ori create <project-name>`

Arguments:

- `project-name` (required) - Name of the project directory

What it does:

1. Creates project directory as sibling to engine
2. Prompts for interactive setup:
    - Game title (display name)
    - Template choice (fresh or platformer)
    - Author name (optional)
    - Description (optional)
3. Copies template files
4. Generates configuration files
5. Sets up project structure

Example:

```
ori create my-platformer

# You'll be prompted:
# Game title: My Awesome Platformer
# Template: 2 (platformer)
# Author: Your Name
# Description: A fun platformer game
```

Output structure:

```
../my-platformer/
├── objects/          # Game object TypeScript files
├── sprites/          # Sprite assets
├── rooms/            # Room JSON files
├── src/              # Entry point (main.ts)
├── game.json         # Game configuration
├── index.html        # HTML entry
├── package.json
└── tsconfig.json
```

Next steps:

```
cd ../my-platformer
npm install
npm run build
# Open index.html in browser
```

`ori update check`

Checks for engine updates without making changes.

Syntax: `ori update check`

What it shows:

- Current version vs latest version
- Changelog of what's new
- Breaking changes warnings

- Compatibility status

Example:

```
ori update check

# Output:
# Current version: 0.1.0
# Latest version: 0.2.0
#
# Changelog:
# - Added audio system
# - Improved collision detection
# - BREAKING: Renamed instance_create_layer to instance_create
#
# Compatibility: Manual migration required
```

Use case: Always run this before updating to see what will change.

`ori update`

Updates the engine to the latest version (engine only, game code unchanged).

Syntax: `ori update`

What it does:

1. Checks for uncommitted changes (blocks if found)
2. Creates automatic backup in `../.bkp/`
3. Fetches latest version from git
4. Rebuilds the engine
5. Updates config files

Example:

```
ori update

# Output:
# Checking for updates...
# Found new version: 0.2.0
# Creating backup...
# ✓ Backup created: ../.bkp/engine-2026-01-29-12-00
# Updating engine...
# ✓ Updated to 0.2.0
# ✓ Rebuilt packages
# ✓ Updated configuration
#
# Update complete!
```

Important: Your game code is NOT modified. Only the engine is updated.

`ori update full`

Updates engine and migrates game code automatically.

Syntax: `ori update full`

What it does:

- Everything from `ori update`
- Plus: Runs migration scripts on your game code
- Updates deprecated function calls
- Adds new required methods
- Updates config files

Example:

```
ori update full

# Output:
# ... (same as ori update)
# Running migrations...
# ✓ Migrated objects/obj_player.ts (renamed 3 functions)
# ✓ Migrated objects/obj_enemy.ts (added lifecycle method)
# ✓ Updated game.json
#
# Migration complete!
# 2 files changed, 5 functions updated
```

When to use: When you want automatic code updates for breaking changes.

`ori dev`

Starts a local development server.

Syntax: `ori dev`

What it does:

- Launches HTTP server on localhost:3000
- Serves your game files
- Provides CORS headers for assets

Example:

```
cd my-game
ori dev

# Output:
# Origami Dev Server
# ==================
# Server running at: http://localhost:3000
# Press Ctrl+C to stop
```

Workflow:

1. Run `ori dev`

2. Open http://localhost:3000 in browser
3. Make changes to TypeScript files
4. Run `npm run build` to recompile
5. Refresh browser

Note: Not currently implemented in all templates. Use `npm start` or open `index.html` directly.

`ori build`

Builds the game for production deployment.

Syntax: `ori build`

What it does:

- Compiles TypeScript to JavaScript
- Copies all assets
- Generates optimized output
- Creates deployment-ready files

Example:

```
ori build

# Output:
# Building Origami Engine project...
# ✓ Compiled TypeScript
# ✓ Copied assets
# ✓ Generated output
# Build complete! Output in: ./build/
```

Output structure:

```
build/
├── js/game.js        # Compiled game code
├── lib/              # Engine runtime
├── sprites/          # Sprite assets
├── rooms/            # Room files
├── index.html
└── game.json
```

Deployment: Upload the `build/` directory to any static host.

Note: Currently, use `npm run build` in project directory. Standalone `ori build` coming soon.

`ori --version`

Shows the installed engine version.

Syntax: `ori --version` or `ori -v`

Example:

```
ori --version
# Origami Engine v0.1.0
```

`ori --help`

Shows help information.

Syntax: `ori --help` or `ori -h` or `ori help`

Example:

```
ori --help

# Output:
# Origami Engine CLI
#
# Usage:
#   ori <command> [options]
#
# Commands:
#   create <name>    Create a new game project
#   update check     Check for engine updates
#   update           Update engine only
#   update full      Update engine + migrate code
#   dev              Start development server
#   build            Build for production
#   --version        Show version
#   --help           Show help
```

Configuration Files

Global Config

Location: `~/.origami/config.json`

Stores engine path and installation info:

```
{
  "enginePath": "D:/Projects/TypeScript/Origami-Engine",
  "version": "0.1.0",
  "installedAt": "2026-01-29T12:00:00.000Z"
}
```

Purpose: Allows `ori` commands to work from any directory.

Engine Config

Location: `.origami/config.json` (in engine root)

Stores engine settings:

```
{
  "version": "0.1.0",
  "lockVersion": false,
  "templateBranches": {
    "fresh": "template/fresh",
    "platformer": "template/platformer"
  },
  "migrations": {}
}
```

Properties:

- `version` - Current engine version
- `lockVersion` - Prevents updates if true
- `templateBranches` - Git branches for templates
- `migrations` - Available migration scripts

Game Config

Location: `game.json` (in game project)

Game-specific settings:

```
{
  "name": "My Platformer",
  "width": 640,
  "height": 480,
  "backgroundColor": "#000000",
  "startRoom": "room_level1"
}
```

Working Directory

Where to Run Commands

`ori create` : Run from engine directory

```
cd origami-engine
ori create my-game
```

`ori update` : Run from anywhere (finds engine via config)

```
cd my-game
ori update check
ori update
```

`npm run build` : Run from game directory

```
cd my-game
npm run build
```

Update Safety Features

Automatic Backups

Every update creates a backup:

```
../.bkp/engine-2026-01-29-12-00/
└── metadata.json  # Contains git commit hash
```

Rollback if needed:

```
cd origami-engine
git reset --hard <commit-from-backup>
pnpm install && pnpm build
```

Uncommitted Changes Check

Update blocks if you have uncommitted changes:

```
ori update
# ❌ You have uncommitted changes in your game project.
#    Please commit or stash them first.
#    Run: git status
```

Version Locking

Lock to specific version in `.origami/config.json`:

```
{
  "lockVersion": true
}
```

Updates will be blocked until unlocked.

Templates

Fresh Template

Empty project structure:

- Minimal setup
- No example objects
- Start from scratch

Platformer Template

Complete working game:

- Player with physics
- Enemies with AI
- Collectibles
- Level design example

Common Workflows

Creating a New Game

```
# 1. Create project
cd origami-engine
ori create my-game

# 2. Navigate and install
cd ../my-game
npm install

# 3. Build and run
npm run build
# Open index.html in browser
```

Updating the Engine

```
# 1. Commit your work
git add .
git commit -m "Save before engine update"

# 2. Check what's new
ori update check

# 3. Update engine (safe)
ori update

# Or update with automatic migration:
ori update full

# 4. Test your game
npm run build
# Open index.html to test
```

Daily Development

```
# 1. Make changes to objects/*.ts
vim objects/obj_player.ts

# 2. Rebuild
npm run build

# 3. Refresh browser to see changes
```

Troubleshooting

"ori: command not found"

Solution 1: Restart terminal **Solution 2**: Run `npm run begin` again to relink CLI **Solution 3**: Use full path: `node /path/to/engine/packages/cli/dist/index.js`

Update Fails

Check:

- ✅ Uncommitted changes committed?
- ✅ Internet connection active?
- ✅ Version not locked?

View backup location to rollback if needed.

Create Fails with Template Error

Make sure template git branches exist:

- `template/fresh`
- `template/platformer`

Next Steps

- **03-creating-games.md** - Using `ori create` in detail
- **12-update-system.md** - Update system deep dive
- **30-cli-commands.md** - Complete CLI reference (advanced)

← Back to Index