CLI Commands Reference

Complete `ori` command reference

Overview

The `ori` CLI provides commands for creating projects, managing updates, and running development servers. Available globally after installation.

Installation

The `ori` command is installed globally during engine setup:

```
cd origami-engine
npm run begin
```

This links the CLI globally, making `ori` available from any directory.

Command Structure

```
ori <command> [options] [arguments]
```

Examples:

```
ori create                    # Create new project
ori dev                       # Start dev server
ori update check              # Check for updates
ori --version                 # Show version
ori --help                    # Show help
```

Global Options

`--version`

Shows the current engine version.

```
ori --version
```

Output:

```
Origami Engine v0.1.0
```

`--help`

Shows help information for all commands.

```
ori --help
```

Output:

```
Origami Engine CLI

Usage: ori <command> [options]

Commands:
  create              Create a new game project
  dev                 Start development server
  build               Build for production
  update check        Check for engine updates
  update              Update engine only
  update full         Update engine and migrate game code

Options:
  --version           Show version number
  --help              Show help
```

Project Commands

`ori create`

Creates a new game project with interactive wizard.

Syntax: `ori create [name] [options]`

Options:

- `--template <name>` - Template to use (fresh, platformer)
- `--name <name>` - Project name
- `--author <name>` - Author name
- `--description <text>` - Project description
- `--no-wizard` - Skip interactive prompts

Interactive Mode (default):

```
ori create
```

Prompts for:

1. Project name
2. Template choice (fresh/platformer)
3. Author name
4. Description

With Arguments:

```
ori create my-game --template platformer --author "John Doe"
```

Non-Interactive:

```
ori create my-game --no-wizard --template fresh
```

Output:

```
✅ Project created: my-game/
✅ Dependencies installed
✅ Ready to develop!

Next steps:
  cd my-game
  ori dev
```

`ori dev`

Starts local development server with hot reload.

Syntax: `ori dev [options]`

Options:

- `--port <number>` - Port number (default: 3000)
- `--host <address>` - Host address (default: localhost)
- `--open` - Open browser automatically

Basic Usage:

```
ori dev
```

Custom Port:

```
ori dev --port 8080
```

Open Browser:

```
ori dev --open
```

Output:

```
🚀 Dev server running at:
   Local:   http://localhost:3000
   Network: http://192.168.1.100:3000

Press Ctrl+C to stop
```

Features:

- Hot reload on file changes
- Automatic TypeScript compilation
- Live error reporting
- Asset watching

`ori build`

Builds project for production deployment.

Syntax: `ori build [options]`

Options:

- `--outdir <path>` - Output directory (default: dist/)
- `--minify` - Minify JavaScript
- `--sourcemap` - Generate source maps

Basic Build:

```
ori build
```

Production Build:

```
ori build --minify
```

With Source Maps:

```
ori build --minify --sourcemap
```

Output:

```
📦 Building project...
✅ TypeScript compiled
✅ Assets bundled
✅ Build complete: dist/

Files:
  dist/index.html          2.3 KB
  dist/bundle.js         156.4 KB
  dist/assets/           12.8 MB
```

Update Commands

`ori update check`

Checks for available engine updates (read-only).

Syntax: `ori update check`

Output:

```
📦 Current version: v0.1.0
🆕 Latest version:  v0.2.0

Changelog:
  - Added particle system
  - Improved collision detection
  - Fixed sprite animation bug

⚠️  Breaking changes:
  - Renamed instance_create() to instance_create_depth()
  - Changed room_goto() signature

Run 'ori update' to update engine only
Run 'ori update full' to update and migrate game code
```

`ori update`

Updates engine to latest version (game code unchanged).

Syntax: `ori update [version] [options]`

Options:

- `--force` - Skip safety checks
- `--no-backup` - Don't create backup

Update to Latest:

```
ori update
```

Update to Specific Version:

```
ori update v0.2.0
```

Process:

1. Check for uncommitted changes
2. Create backup in `../.bkp/`
3. Fetch latest version from git
4. Install dependencies
5. Build packages
6. Update config files

Output:

```
🔄 Checking for updates...
✅ Found v0.2.0

🔄 Creating backup...
✅ Backup created: ../.bkp/engine-2026-01-29-12-00/

🔄 Updating engine...
✅ Checked out v0.2.0
✅ Dependencies installed
✅ Packages built
✅ Config updated

✅ Update complete!

Your game code is unchanged.
Run 'ori update full' to migrate game code.
```

`ori update full`

Updates engine AND migrates game code automatically.

Syntax: `ori update full [version] [options]`

Options:

- `--force` - Skip safety checks
- `--no-backup` - Don't create backup

Full Update:

```
ori update full
```

Process:

1. All steps from `ori update`
2. Run AST-based migrations on game code
3. Display migration report

Output:

```
🔄 Checking for updates...
✅ Found v0.2.0

🔄 Creating backup...
✅ Backup created: ../.bkp/engine-2026-01-29-12-00/

🔄 Updating engine...
✅ Checked out v0.2.0
✅ Dependencies installed
✅ Packages built
✅ Config updated

🔄 Running migrations...
✅ Renamed instance_create() → instance_create_depth()
✅ Updated room_goto() calls
✅ Added new lifecycle methods

Migration Report:
  objects/obj_player.ts      3 changes
  objects/obj_enemy.ts       1 change
  objects/obj_bullet.ts      2 changes

✅ Full update complete!

Test your game to verify changes:
  ori dev
```

Utility Commands

`ori rollback`

Rolls back to previous engine version (uses last backup).

Syntax: `ori rollback [backup]`

Rollback to Last Backup:

```
ori rollback
```

Rollback to Specific Backup:

```
ori rollback ../.bkp/engine-2026-01-28-10-30/
```

Output:

```
🔄 Rolling back to v0.1.0...
✅ Restored from backup
✅ Dependencies reinstalled
✅ Packages rebuilt

✅ Rollback complete!
```

`ori list-backups`

Lists all available backups.

Syntax: `ori list-backups`

Output:

```
Available backups:

  engine-2026-01-29-12-00/
  Version: v0.2.0
  Created: 2026-01-29 12:00:00

  engine-2026-01-28-10-30/
  Version: v0.1.0
  Created: 2026-01-28 10:30:00

To restore a backup:
  ori rollback <backup-name>
```

`ori config`

Shows current engine configuration.

Syntax: `ori config [options]`

Options:

- `--global` - Show global config (~/.origami/config.json)
- `--engine` - Show engine config (.origami/config.json)
- `--game` - Show game config (game.json)

Show All Configs:

```
ori config
```

Show Global Config:

```
ori config --global
```

Output:

```
Global Config (~/.origami/config.json):
  Engine Path: D:/Projects/TypeScript/Origami-Engine
  Version:     v0.1.0
  Installed:   2026-01-29T12:00:00.000Z

Engine Config (.origami/config.json):
  Version:      v0.1.0
  Lock Version: false
  Templates:    fresh, platformer

Game Config (game.json):
  Name:        My Platformer
  Version:     1.0.0
  Author:      John Doe
  Entry Room:  rm_level1
```

Error Handling

Common Errors

Not in a Game Project:

```
ori dev
```

```
❌ Not in a game project
   Create one with: ori create
```

Version Locked:

```
ori update
```

```
❌ Version is locked. Updates are disabled.
   To unlock: Edit .origami/config.json and set "lockVersion": false
```

Uncommitted Changes:

```
ori update
```

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

Update Failed:

```
ori update
```

```
❌ Update failed: Failed to fetch git tags
   Check your internet connection.

🔄 Rolling back...
✅ Rollback complete. Engine restored to previous state.
```

Exit Codes

- `0` - Success
- `1` - General error
- `2` - Invalid arguments
- `3` - Not in project directory
- `4` - Network error
- `5` - Update/migration failed

Example Usage in Scripts:

```
ori update
if [ $? -eq 0 ]; then
  echo "Update successful"
else
  echo "Update failed"
fi
```

Environment Variables

`ORIGAMI_ENGINE_PATH`

Override global engine path.

```
export ORIGAMI_ENGINE_PATH=/path/to/engine
ori dev
```

`ORIGAMI_NO_COLOR`

Disable colored output.

```
export ORIGAMI_NO_COLOR=1
ori build
```

`ORIGAMI_LOG_LEVEL`

Set logging verbosity (debug, info, warn, error).

```
export ORIGAMI_LOG_LEVEL=debug
ori update
```

Configuration Files

Global Config

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

Format:

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

Engine Config

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

Format:

```
{
  "version": "0.1.0",
  "lockVersion": false,
  "templateBranches": {
    "fresh": "template/fresh",
    "platformer": "template/platformer"
  },
  "migrations": {
    "0.2.0": ["rename-api-functions", "add-lifecycle-methods"]
  }
}
```

Game Config

Location: `game.json` (game root)

Format:

```
{
  "name": "My Platformer",
  "version": "1.0.0",
  "author": "John Doe",
  "description": "A fun platformer game",
  "entryRoom": "rm_level1",
  "engineVersion": "0.1.0"
}
```

Next Steps

- **31-cli-configuration.md** - Configuration options
- **32-cli-templates.md** - Template system
- **11-cli-reference.md** - Quick reference

← Back to Index