ORM Tools

Prisma CLI

The Prisma command line interface (CLI) is the primary way to interact with your Prisma project from the command line. It can initialize new project assets, generate Prisma Client, and analyze existing database structures through introspection to automatically create your application models.

Command reference

See Prisma CLI command reference for a complete list of commands.

Installation

The Prisma CLI is typically installed locally as a **development dependency**, that's why the `--save-dev` (npm) and `--dev` (Yarn) options are used in the commands below.

> **INFO**
>
> We **recommend that you install the Prisma CLI locally** in your project's `package.json` to avoid version conflicts that can happen with a global installation.

npm

Install with npm:

```
npm install prisma --save-dev
```

Yarn

Install with yarn:

pn

Inst

Ask AI

```
pnpm install prisma --save-dev
```

Bun

Install with Bun:

```
bun add prisma
```

Global installation (Not recommended)

Usage

If you installed Prisma as a development dependency, you need to prefix the `prisma` command with your package runner.

npm

```
npx prisma
```

Yarn

```
yarn prisma
```

pnpm

```
pnpm dlx prisma
```

Bun

```
bunx prisma
```

Sy

The installed. When called without argu nt:

```
$
```

Show CLI results

```
Prisma is a modern DB toolkit to query, migrate and model your database (https://www.prisma.

Usage

  $ prisma [command]

Commands

            init   Setup Prisma for your app
        generate   Generate artifacts (e.g. Prisma Client)
              db   Manage your database schema and lifecycle
         migrate   Migrate your database
          studio   Browse your data with Prisma Studio
        validate   Validate your Prisma schema
          format   Format your Prisma schema

Flags

     --preview-feature   Run Preview Prisma commands

Examples

  Setup a new Prisma project
  $ prisma init

  Generate artifacts (e.g. Prisma Client)
  $ prisma generate

  Browse your data
  $ prisma studio

  Create migrations from your Prisma schema, apply them to the database, generate artifacts
  $ prisma migrate dev

  Pull the schema from an existing database, updating the Prisma schema
  $ prisma db pull

  Push the Prisma schema state to the database
  $ prisma db push
```

You can get additional help on any of the `prisma` commands by adding the `--help` flag after the con

Ex

All ey exit:

- exit code 0 when a command runs successfully

- exit code 1 when a command errors
- exit code 130 when the CLI receives a signal interrupt (SIGINT) message or if the user cancels a prompt. This exit code is available in Prisma ORM versions 4.3.0 and later.

Telemetry

The term **telemetry** refers to the collection of certain usage data to help *improve the quality of a piece of software*. Prisma uses telemetry in two contexts:

- when it collects CLI usage data
- when it submits CLI error reports

This page describes the overall telemetry approach for Prisma, what kind of data is collected and how to opt-out of data collection.

Why does Prisma collect metrics?

Telemetry helps us better understand *how many users* are using our products and *how often* they are using our products. Unlike many telemetry services, our telemetry implementation is intentionally limited in scope and is actually useful for the developer:

- **Limited in scope**: We use telemetry to answer one question: how many monthly active developers are using Prisma CLI?
- **Provides value**: Our telemetry service also checks for version updates and offers security notices.

When is data collected?

Data is collected in two scenarios that are described below.

Usage data

Invocations of the `prisma` CLI and general usage of Studio results in data being sent to the telemetry server at https://checkpoint.prisma.io. Note that:

- The data does **not** include your schema or the data in your database
- Prisma only sends information after you execute a CLI command

Her

		)
version	*string*	Currently installed version of the product (e.g. `1.0.0-rc0`)

Field	Attributes	Description
`arch`	*string*	Client's operating system architecture (e.g. `amd64`).
`os`	*string*	Client's operating system (e.g. `darwin`).
`node_version`	*string*	Client's node version (e.g. `v12.12.0`).
`signature`	*string*	Random, non-identifiable signature UUID (e.g. `91b014df3-9dda-4a27-a8a7-15474fd899f8`)
`user_agent`	*string*	User agent of the checkpoint client (e.g. `prisma/js-checkpoint`)
`timestamp`	*string*	When the request was made in RFC3339 format (e.g. `2019-12-12T17:45:56Z`)

You can opt-out of this behavior by setting the `CHECKPOINT_DISABLE` environment variable to `1`, e.g.:

```
$ export CHECKPOINT_DISABLE=1
```

Error reporting

Prisma potentially collects error data when there is a crash in the CLI.

Before an error report is submitted, there will *always* be a prompt asking you to confirm or deny the submission of the error report! Error reports are never submitted without your explicit consent!

How to opt-out of data collection?

Usage data

You can opt-out of usage data collection by setting the `CHECKPOINT_DISABLE` environment variable to `1`, e.g.:

```
$ export CHECKPOINT_DISABLE=1
```

Err

You active prompt with *no*.

Edit