软件成分分析命令行工具

使用手册

# **功能模块说明**

软件成分分析模块为一个面向开源软件的依赖成分检测与分析命令行工具。其将项目源文件作为输入，通过对项目的编译和对特定配置文件的解析，检测项目中所使用的第三方库，并生成检测结果。该模块基于Dependency Check和Scancode等开源项目开发，具有良好的可扩展性，准确率高，可在Linux操作系统中搭建运行。

表 1 检测功能支持的**配置文件(Files)**类型与所需的**编译环境(Executables)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type | Name(34) | Language(20) | Files | Executables |
| CARGO | Cargo | Rust | Cargo.toml, Cargo.lock | cargo |
| CARTHAGE | Carthage | Objective C | Cartfile, Cartfile.resolved |  |
| CLOJARS | Leiningen | Clojure | project.clj | lein |
| COCOAPODS | Pod Lock | Objective C | .podspec,  Podfile, Podfile.lock |  |
| COMPOSER | Composer | PHP | composer.json, composer.lock | composer |
| CONAN | Conan | C/C++ | conanfile.py, conan.lock | conan |
| CONDA | Conda Cli | Python | environment.yml |  |
| CPAN | Cpan Cli | Perl | Makefile.PL, Build.PL | cpanm |
| CPAN | Cpan Deps | Perl | cpanfile |  |
| CRAN | Cran Deps | R | DESCRIPTION |  |
| CRAN | Packrat Lock | R | packrat.lock |  |
| DART | Dart Pub | Dart | pubspec.yaml, pubspec.lock | dart, flutter |
| DEP | Go Lock | Golang | Gopkg.lock |  |
| GO\_DEP | Go Dep | Golang | Godeps.json |  |
| GO\_MOD | Go Mod Cli | Golang | go.mod, go.sum | go |
| GRADLE | Gradle | Java | build.gradle | gradle |
| HACKAGE | Cabal | Haskell | <package\_name>.cabal |  |
| HACKAGE | Stack | Haskell | stack.yaml, package.yaml | stack |
| HEX | Mix | Elixir | mix.exs, mix.lock | mix |
| HEX | Rebar | Erlang | rebar.config | rebar3 |
| MAVEN | Maven Pom | Java | pom.xml | mvn |
| NPM | Npm Cli | Node JS | package.json, package-lock.json | npm |
| NPM | Shrinkwrap | Node JS | package.json,  npm-shrinkwrap.json |  |
| NUGET | Project | C# | .csproj, .nuspec |  |
| NUGET | Nugetconf | C# | packages.config |  |
| PIP | Pip Env | Python | Pipfile, Pipfile.lock |  |
| PIP | Pip Inspector | Python | setup.py, requirements.txt |  |
| PNPM | Pnpm Lock | Node JS | package.json,  pnpm-lock.yaml |  |
| POETRY | Poetry | Python | pyproject.toml, poetry.lock |  |
| RUBYGEMS | Gemlock | Ruby | .gemspec, Gemfile,  Gemfile.lock | gem, bundler |
| SCALA | Sbt | Scala | build.sbt | sbt,  sbt-dependency-gragh |
| SCALA | Scala Deps | Scala | Dependencies.scala |  |
| SWIFT | Swift | Swift | Package.swift, Package.resolved | swift |
| YARN | Yarn Lock | Node JS | package.json, yarn.lock |  |

功能模块目前拥有独立的检测器（detector）**34种**，支持检测的语言（language）**20种**，分别是：C, C++, C#, Clojure, Dart, Elixir, Erlang, Golang, Haskell, Java, Node JS, Objective C, Perl, PHP, Python, R, Ruby, Rust, Scala, Swift. 各种语言的检测流程见下表。

表 2 检测功能支持的**包管理器(Package Manager)或语言(Language)**类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Package Manager/Language | Buildless | Build | Process | 说明 |
| Cargo Support  (Language: Rust) | √ | √ | **Buildless mode:**  解析Cargo.lock文件，如果没有Cargo.lock文件，则解析Cargo.toml文件  **Build mode:**  调用cargo编译命令，根据Cargo.toml文件生成Cargo.lock文件并解析 | 项目最好含有Cargo.lock;  建议设置search\_depth为项目目录层数 |
| Composer Support  (Language: PHP) | √ | √ | **Buildless mode:**  解析composer.lock文件，如果没有composer.lock文件，则解析composer.json文件  **Build mode:**  调用composer编译命令，根据composer.json文件生成composer.lock文件并解析 | 项目最好含有composer.lock;  建议设置search\_depth为项目目录层数 |
| Conan Support  (Language: C/C++) | √ | √ | **Buildless mode:**  解析conan.lock文件  **Build mode:**  调用conan编译命令，根据conanfile.py文件生成conan.lock文件并解析 | 项目最好含有conan.lock;  **Build**模式下，项目必须含有conanfile.py;  建议设置search\_depth为项目目录层数 |
| Cpan Support  (Language: Perl) | √ | √ | **Buildless mode:**  解析cpanfile文件  **Build mode:**  调用cpanm编译命令，根据Makefile.PL或Build.PL文件生成依赖树cpan\_deps.txt文件并解析 | 项目最好含有cpanfile;  **Build**模式下，项目必须含有Makefile.PL或Build.PL;  建议设置search\_depth为项目目录层数 |
| Cran Support  (Language: R) | √ | × | **Buildless mode:**  2种配置文件解析器：   1. Cran Deps Detector   解析DESCRIPTION文件   1. Packrat Lock Detector   解析packrat.lock文件 | 项目最好含有packrat.lock;  建议设置search\_depth为项目目录层数 |
| Dart Pub Support  (Langugage: Dart) | √ | √ | **Buildless mode:**  解析pubspec.lock文件，如果没有pubspec.lock文件，则解析pubspec.yaml文件  **Build mode:**  调用dart/flutter编译命令，根据pubspec.yaml文件生成pubspec.lock文件并解析 | 项目最好含有pubspec.lock;  建议设置search\_depth为项目目录层数 |
| Golang Support | √ | √ | **Buildless mode:**  3种配置文件解析器：   1. Go Dep Detector   解析Godeps.json文件   1. Go Lock Detector   解析Gopkg.lock文件   1. Go Mod Detector   解析go.sum文件，如果没有go.sum文件，则解析go.mod文件  **Build mode:**  调用go编译命令，根据go.mod和go.sum文件生成依赖列表gomod\_list.txt文件并解析 | **Build**模式下，项目必须同时含有go.mod和go.sum;  建议设置search\_depth为项目目录层数 |
| Gradle Support  (Language: Java) | √ | √ | **Buildless mode:**  解析build.gradle文件  **Build mode:**  调用gradle编译命令，根据build.gradle文件生成依赖树gradle\_tree.txt文件并解析 | 项目必须含有build.gradle;  建议设置search\_depth为项目目录层数 |
| Haskell Support | √ | √ | **Buildless mode:**  2种配置文件解析器：   1. Package Cabal Detector   解析<package\_name>.cabal文件   1. Package Stack Detector   解析package.yaml文件  **Build mode:**  调用stack编译命令，根据stack.yaml文件生成依赖关系stack\_deps.json文件并解析 | 项目最好含有stack.yaml;  建议设置search\_depth为项目目录层数 |
| Lein Support  (Language: Clojure) | √ | √ | **Buildless mode:**  解析project.clj文件  **Build mode:**  调用lein&mvn编译命令，根据project.clj文件生成依赖树lein\_maven\_tree.txt文件并解析 | 项目必须含有project.clj;  建议设置search\_depth为项目目录层数 |
| Maven Support  (Language: Java) | √ | √ | **Buildless mode:**  解析pom.xml文件  **Build mode:**  调用mvn编译命令，根据pom.xml文件生成依赖树maven\_tree.txt文件并解析 | 项目必须含有pom.xml;  建议设置search\_depth为项目目录层数 |
| Mix Support  (Language: Elixir) | √ | √ | **Buildless mode:**  解析mix.lock文件，如果没有mix.lock文件，则解析mix.exs文件  **Build mode:**  调用mix编译命令，根据mix.exs文件生成mix.lock文件并解析 | 项目最好含有mix.lock;  建议设置search\_depth为项目目录层数 |
| NodeJS Support | √ | √ | **Buildless mode:**  5种配置文件解析器：  解析package-lock.json, npm-shrinkwrap.json, pnpm-lock.yaml和yarn.lock四种锁文件中的一种，如果没有任何一种锁文件，则解析package.json文件  **Build mode:**  调用npm编译命令，根据package.json文件生成package-lock.json文件并解析 | 项目必须含有package.json |
| Nuget Support  (Language: C#) | √ | × | **Buildless mode:**  2种配置文件解析器：   1. Project Detector   解析.csproj,  .nuspec文件   1. Nugetconf Detector   解析packages.config文件 | 建议设置search\_depth为项目目录层数 |
| Objective C Support | √ | × | **Buildless mode:**  2种配置文件解析器：   1. Cocoapods Detector   解析Podfile.lock文件，如果没有Podfile.lock文件，则解析Podfile或<project\_name>.podspec文件   1. Carthage Detector   解析Cartfile.resolved文件，如果没有Cartfile.resolved文件，则解析Cartfile文件 | 建议设置search\_depth为项目目录层数 |
| Python Support | √ | × | **Buildless mode:**  4种配置文件解析器：   1. Conda Detector   解析environment.yml文件   1. Pip Detector   解析setup.py和requirements.txt文件   1. Pipenv Detector   解析Pipfile.lock文件，如果没有Pipfile.lock文件，则解析Pipfile文件   1. Poetry Detector   解析poetry.lock文件，如果没有poetry.lock文件，则解析pyproject.toml文件 | 建议设置search\_depth为项目目录层数 |
| Rebar Support  (Language: Erlang) | √ | √ | **Buildless mode:**  解析rebar.config文件  **Build mode:**  调用rebar3编译命令，根据rebar.config文件生成依赖树rebar\_tree.txt文件并解析 | 项目必须含有rebar.config;  建议设置search\_depth为项目目录层数 |
| Ruby Support | √ | √ | **Buildless mode:**  解析Gemfile.lock文件，如果没有Gemfile.lock文件，则解析Gemfile文件或<project\_name>.gemspec文件  **Build mode:**  调用bundler编译命令，根据Gemfile文件生成Gemfile.lock文件并解析 | 项目最好含有Gemfile.lock;  建议设置search\_depth为项目目录层数 |
| Scala Support | √ | √ | **Buildless mode:**  2种配置文件解析器：   1. Sbt Detector   解析build.sbt文件   1. Scala Deps Detector   解析Dependencies.scala  文件  **Build mode:**  调用sbt编译命令，根据build.sbt文件生成依赖关系树tree.json文件并解析 | **Build**模式下，项目必须含有build.sbt;  建议设置search\_depth为项目目录层数 |
| Swift Support | √ | √ | **Buildless mode:**  解析Package.resolved文件，如果没有Package.resolved文件，则解析Package.swift文件  **Build mode:**  调用swift编译命令，根据Package.swift文件生成Package.resolved文件并解析 | 项目最好含有Package.resolved;  建议设置search\_depth为项目目录层数 |

# **功能模块安装**

本功能模块采用Python语言开发，基于Dependency Check和Scancode等开源工具扩展，可运行在Linux平台上，环境所需工具均为开源软件，所依赖的第三方库均为开源库。

* 操作系统要求：Linux Ubuntu 20.04.2
* 运行环境要求：

Python 3.8.10, pip 20.0.2, appdirs 1.4.4, bottle 0.12.23, certifi 2021.10.8, chardet 5.0.0, charset-normalizer 2.1.1, colorama 0.4.4, commonmark 0.9.1, conan 1.54.0, dparse2 0.6.0, fasteners 0.18, filelock 3.6.0, future 0.18.2, gemfileparser 0.8.0, idna 3.4, Jinja2 3.1.2, MarkupSafe 2.1.1, node-semver 0.6.1, packaging 21.3, patch-ng 1.17.4, pip-requirements-parser 31.2.0, pluginbase 1.0.1, pygmars 0.7.0, Pygments 2.12.0, PyJWT 2.6.0, pyparsing 3.0.9, python-dateutil 2.8.2, PyYAML 6.0, requests 2.28.1, six 1.16.0, toml 0.10.2, tqdm 4.64.1, urllib3 1.26.12

* 编译环境要求：

表 3 功能模块编译环境所需的软件

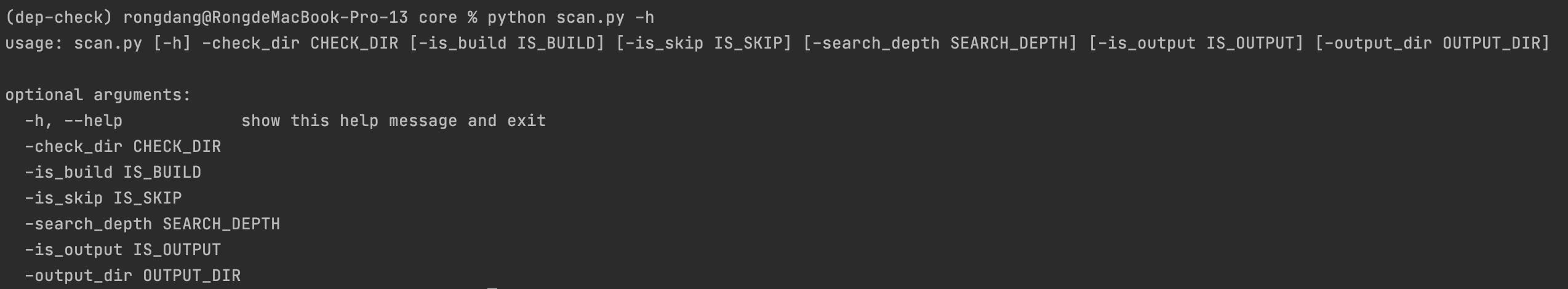
|  |  |  |  |
| --- | --- | --- | --- |
| **软件名称** | **软件版本** | **相关说明** | **包管理器** |
| Java Virtual Machine | 11.0.1 | JVM | × |
| Apache Maven | 3.8.6 | 依赖于JVM | √ |
| Gradle | 6.8.2 | 依赖于JVM | √ |
| Leiningen | 2.9.10 | 依赖于JVM | √ |
| sbt | 1.7.3 | 依赖于JVM | √ |
| sbt-dependency-graph | latest | 依赖于sbt | × |
| Node | 17.9.0 |  | × |
| npm | 8.5.5 | 依赖于Node | √ |
| go | 1.19.3 |  | √ |
| PHP | 7.4.3 |  | × |
| composer | 2.4.4 | 依赖于PHP | √ |
| cargo | 1.61.0 |  | √ |
| perl | 5.30.0 |  | × |
| cpanm | 1.7046 | 依赖于Perl 5 | √ |
| ruby | 2.7.0p0 |  | × |
| gem | 3.1.2 | 依赖于ruby | √ |
| Bundler | 2.3.25 | 依赖于gem | √ |
| Flutter SDK | 3.0.5 | 包含dart 2.17.6 | √ |
| swift | 5.7 |  | √ |
| Erlang/OTP | 25.1 |  | × |
| rebar | 3.20.0 | 依赖于Erlang/OTP | √ |
| Elixir | 1.13.4 | 依赖于Erlang/OTP | × |
| mix | 1.13.4 | 依赖于Elixir | √ |
| stack | 2.9.1 |  | √ |

* 网络环境要求：10M带宽

# **功能模块使用**

## **功能模块运行**

在sca-2.0/core目录下，输入python scan.py -h获取帮助信息，结果如下：



**图 1 获取帮助信息**

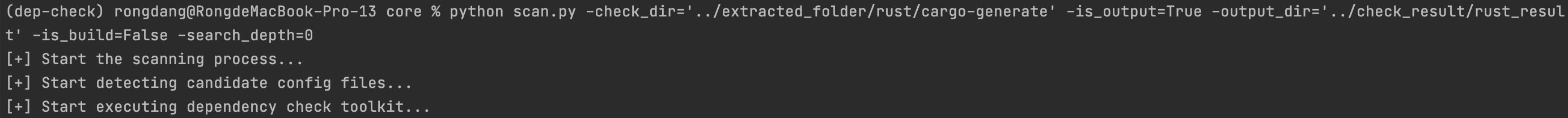
表 4 命令行工具调用参数说明

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名称 | 参数类型 | 是否必选 | 参数说明 |
| check\_dir | string | 是 | 待测项目所在路径 |
| is\_build | bool | 否 | 是否编译构建项目  默认为False |
| is\_skip | bool | 否 | 是否跳过devDependencies  默认为False |
| search\_depth | int | 否 | 搜索深度  默认为3 |
| is\_output | bool | 否 | 是否以JSON文件形式输出检测结果  默认为False |
| output\_dir | string | 否 | 检测结果所在路径  默认为  ‘../check\_result’ |

其他说明：

* check\_dir为项目文件所在路径**（建议使用绝对路径）**，**不支持压缩文件检测**。
* is\_build仅针对Bundler, Cargo, Composer, Conan, Cpan, Dart, Go, Gradle, Leiningen, Maven, Mix, NPM, Rebar3, Sbt, Stack和Swift管理的项目可以设置为True。
* is\_skip仅针对Composer, Dart和NPM管理的项目可进行设置。
* 当前检测功能仅支持Gradle管理的Java项目，不支持Andorid项目。

使用命令行工具对项目进行检测，调用命令示例如下：

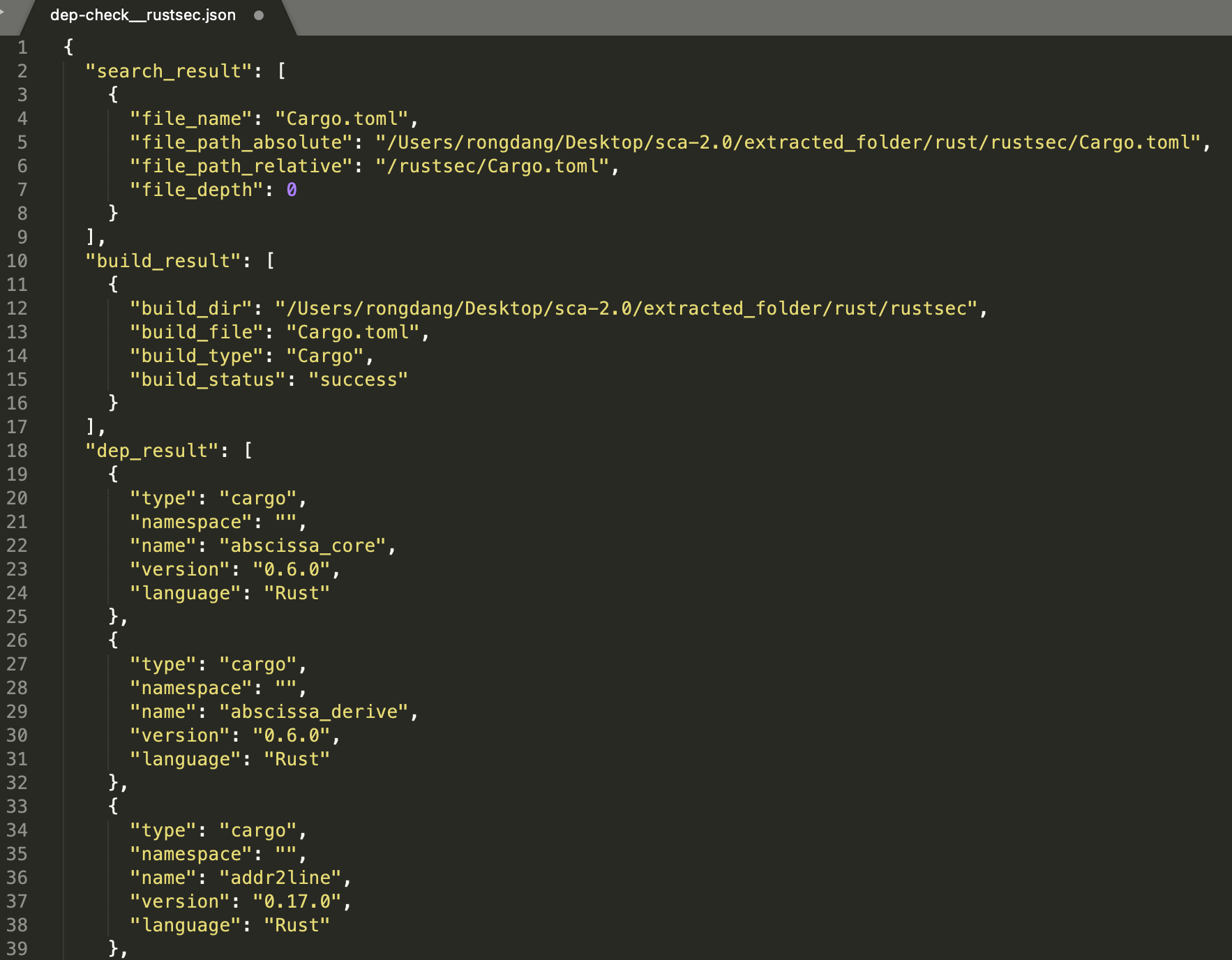


**图 2 命令行工具调用示例**

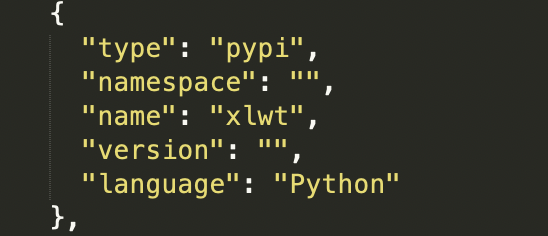
输入参数可有6个，分别是：1) 项目文件路径-check\_dir, 2) 是否输出-is\_output, 3) 检测结果路径-output\_dir, 4) 是否构建-is\_build, 5) 是否跳过devDependencies -is\_skip, 6) 搜索深度-search\_depth.

## **3.2 检测结果说明**

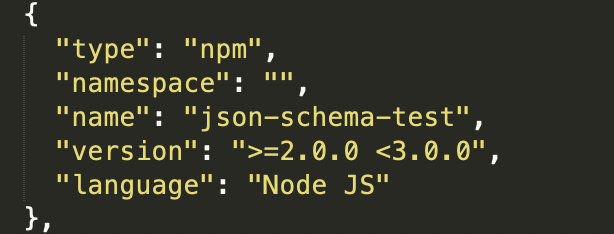
项目检测完成后，若选择以JSON文件形式输出检测结果，则文件内容示例如下：



**图 3 检测结果JSON文件示例**



**图 4 检测结果dep\_result-示例1**



**图 5 检测结果dep\_result-示例2**

字段说明：

dep\_result列表（项目的依赖项条目列表）包含的字段及说明如下：

* type: 包管理器类型 (required)
* namespace: 依赖项名称前缀 (optional)
* name: 依赖项名称 (required)
* version: 依赖项版本 (optional)
* language: 依赖项语言 (required)

表 5 项目的依赖项条目列表元素属性描述

|  |  |
| --- | --- |
| 属性名称 | 描述 |
| type | 包管理器类型 |
| namespace | 命名空间（依赖项名称前缀） |
| name | 依赖项名称 |
| version | 依赖项版本（见示例1, 示例2 ‘version’） |
| language | 依赖项语言 |

项目检测完成后，若不选择以JSON文件形式输出检测结果，则返回字段如下：

表 6 检测功能函数返回字段及描述

|  |  |  |
| --- | --- | --- |
| 字段名称 | 字段类型 | 描述 |
| success | bool | 检测成功(True)  或检测失败(False) |
| result | list | 依赖项条目列表(详见表4) |
| message | string | 检测结果描述(Succeed/Partial Succeed/Failure + Error\_Msg) |