PyInstaller介绍

PyInstaller 是一个跨平台的工具，它将 Python 脚本打包成独立的可执行文件（如 Windows 上的 .exe ，或 Linux/macOS 上的可执行文件）。它可以将一个 Python 程序及其依赖的所有库、资源和外部文件一起打包，使得最终用户无需安装 Python 环境即可运行该程序。PyInstaller 是一个非常有用的工具，尤其是在需要将 Python 程序分发给没有安装 Python 的用户时。

1. PyInstaller核心特点

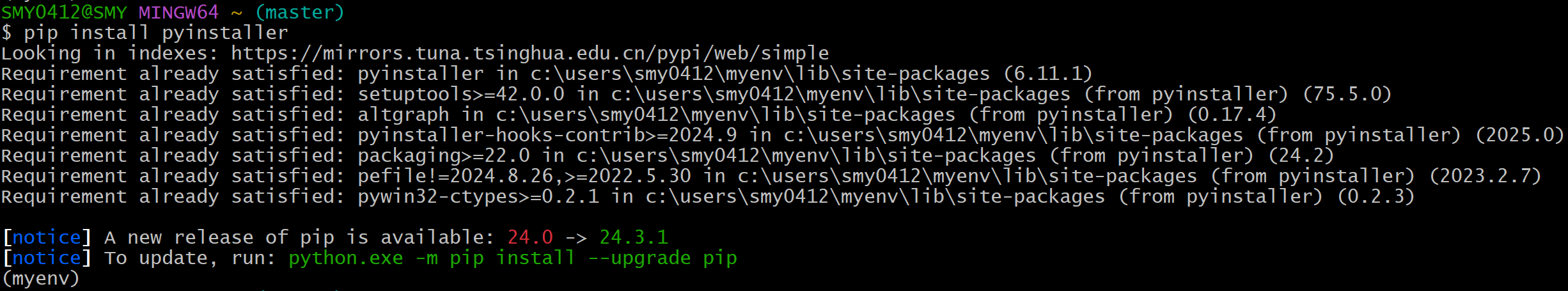
1. 跨平台支持：PyInstaller 支持多个操作系统，包括 Windows、Linux 和 macOS。你可以在任何一个平台上将程序打包成适用于目标平台的可执行文件。
2. 支持 Python 版本：PyInstaller 支持 Python 2.7 和 Python 3.x 版本。
3. 自动检测依赖：它会自动检测并打包所有的 Python 库和模块。
4. 支持大型程序：对于大型程序，PyInstaller 会将所有依赖和资源打包成一个文件，使得分发和运行变得更加简便。
5. GUI 支持：支持将图形界面（如使用 PyQt 或 Tkinter）应用打包为可执行文件。
6. 简便的命令行界面：通过简单的命令行命令，你可以将 Python 脚本转化为可执行文件。

2. PyInstaller打包过程

2.1安装 PyInstaller

首先，你需要在你的 Python 环境中安装 PyInstaller：

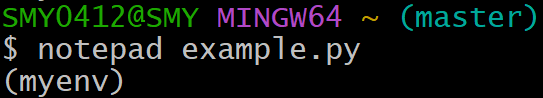
pip install pyinstaller



2.2简单示例程序

建立一个简单的 Python 脚本 example.py：

notepad example.py



其内容如下：

# example.py

import sys

def greet():

print("Hello, World!")

print(f"Arguments passed: {sys.argv}")

if \_\_name\_\_ == "\_\_main\_\_":

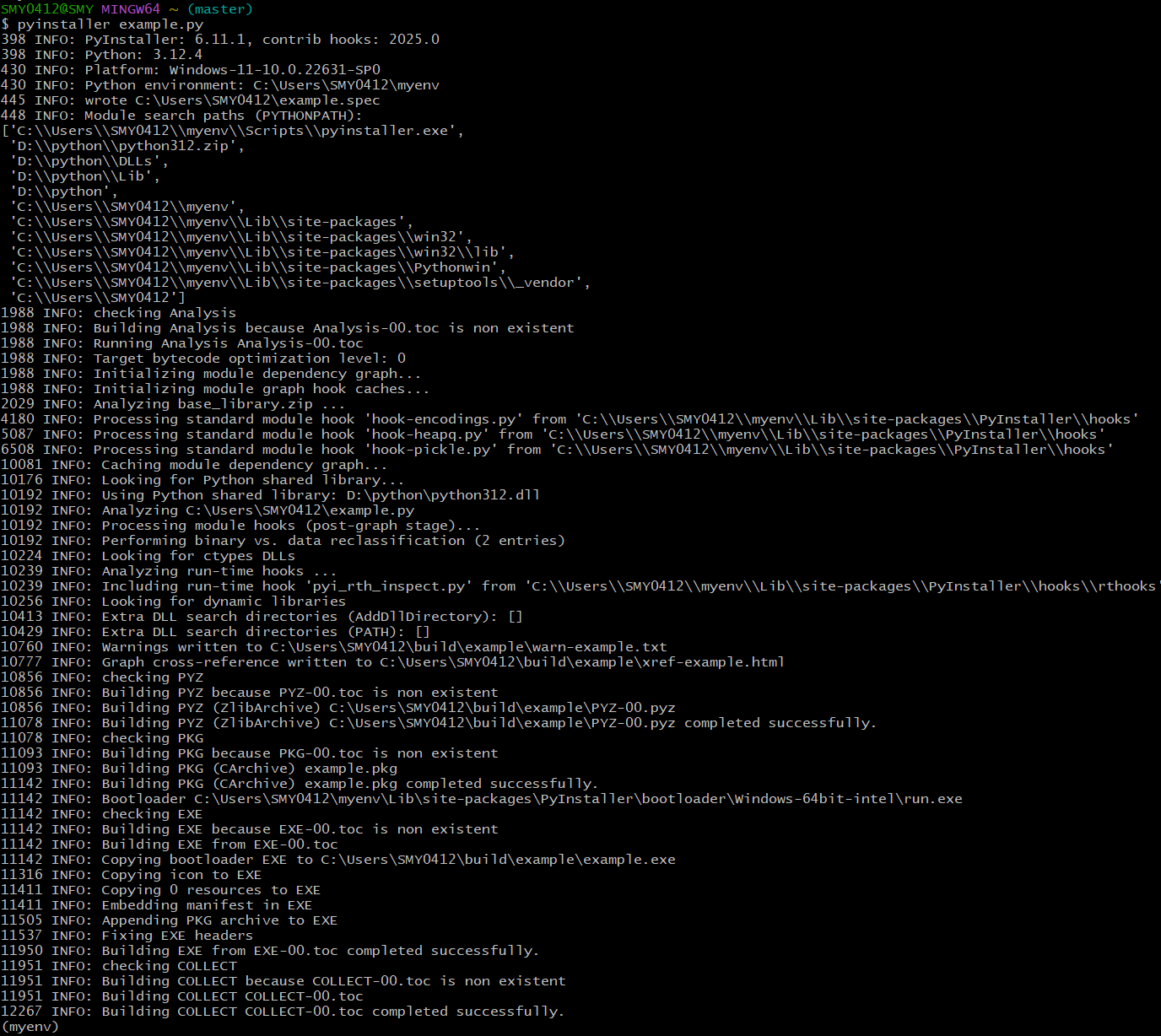
greet()

2.3打包 Python 脚本

在终端（命令行）中将 example.py 打包成一个独立的可执行文件：

pyinstaller example.py

执行该命令后，PyInstaller 会分析Python 脚本，识别它所依赖的模块，并将它们一起打包。



2.3.1打包过程的目录结构

运行命令后，PyInstaller 会生成一个新的目录 dist，其中包含打包好的可执行文件（对于 Windows 是 .exe 文件，Linux 和 macOS 会生成相应的二进制文件）。

PyInstaller 会创建以下目录结构：

project/

│

├── build/ # 临时文件存放目录

│

├── dist/ # 最终的可执行文件目录

│ └── example/ # 可执行文件所在的目录

│ └── example.exe # 可以直接运行的可执行文件

│

├── example.py # 原始的 Python 脚本

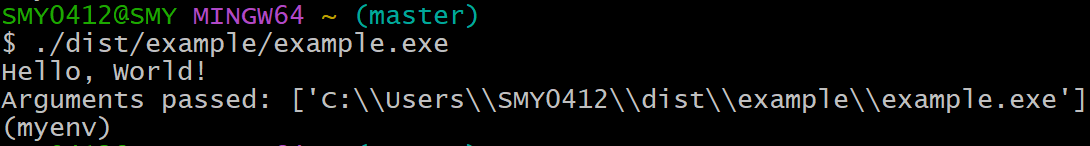
└── example.spec # PyInstaller 配置文件（自动生成）

* **build/** 目录包含构建过程中的临时文件，通常可以忽略。
* **dist/** 目录是需要重点关注的地方，它包含了打包好的可执行文件。
* **example.spec** 文件是 PyInstaller 根据脚本生成的配置文件，可以编辑它来自定义打包过程。

2.3.2可执行文件的运行

可以直接在终端中运行生成的可执行文件，或者双击它，程序会启动并显示相应的输出：

./dist/example/example.exe



2.4常见的打包选项

PyInstaller 提供了许多选项，允许定制打包过程。

2.4.1打包为单个文件

默认情况下，PyInstaller 会将依赖项和脚本打包成多个文件。如果想将所有内容打包成一个单独的可执行文件，可以使用 --onefile 选项：

pyinstaller --onefile example.py

这将把所有依赖的文件打包到一个可执行文件中，但会导致首次启动时的启动时间稍长，因为它需要临时解压文件。

2.4.2禁用控制台窗口（适用于 GUI 应用）

如果程序是一个 GUI 应用，不想在运行时看到控制台窗口，可以使用 --windowed 选项：

pyinstaller --onefile --windowed example.py

这个选项会避免在启动时弹出命令行窗口，适用于使用 Tkinter、PyQt 等库的图形界面应用。

2.4.3自定义图标

如果想为生成的可执行文件指定一个图标，可以使用 --icon 选项：

pyinstaller --onefile --icon=icon.ico example.py

这样生成的可执行文件将带有你提供的图标。

2.4.4添加数据文件

如果需要附带一些额外的文件（例如配置文件、图片、数据库等），可以在打包时通过 --add-data 选项指定：

pyinstaller --onefile --add-data "data/config.json:." example.py

这会将 config.json 文件添加到生成的可执行文件中，并且可以通过相对路径进行访问。格式是 "source:destination"，注意在 Windows 上使用分号（;）作为分隔符，Linux 或 macOS 上使用冒号（:）。

2.4.5使用 spec 文件

PyInstaller 会生成一个 .spec 文件，该文件记录了打包过程的所有设置。可以通过修改 .spec 文件来自定义打包过程，然后运行以下命令来使用它：

pyinstaller example.spec

这个方式非常灵活，通常在需要执行更复杂的打包操作时使用。

2.5常见问题解决

2.5.1依赖问题

有时，某些库可能没有被正确地打包，导致生成的可执行文件在没有 Python 环境的机器上无法运行。

在 --onefile 模式下，PyInstaller 会尝试自动分析并包括所有依赖。如果有遗漏，你可以手动编辑 .spec 文件或使用 --hidden-import 选项添加缺失的依赖。

例如，某些动态导入的模块可能会被遗漏，可以用以下命令添加：

pyinstaller --onefile --hidden-import=module\_name example.py

2.5.2程序启动慢

如果打包成一个单文件可执行文件（--onefile）时程序启动较慢，PyInstaller 会在启动时解压文件到临时目录。这通常是正常现象，但是可以尝试使用 --no-patched 或 --strip 来优化启动时间。

2.5.3操作系统兼容性

在 Windows 上生成的 .exe 文件不能直接在 Linux 或 macOS 上运行，反之亦然，必须在相应操作系统上生成可执行文件。

3. PyInstaller总结

PyInstaller 是一个非常强大的工具，可以将 Python 程序打包成独立的可执行文件，支持多个平台和 Python 版本。它能自动检测依赖并将所有文件打包，使得用户可以在没有安装 Python 环境的情况下运行应用程序。

通过 PyInstaller，可以非常方便地将 Python 脚本转化为可以分发的可执行文件，尤其适合需要将程序提供给非技术用户或在没有 Python 环境的机器上运行时。