**操作流程**

拉取代码

```
git clone <项目仓库地址>
cd <项目目录>
```

安装 Poetry（197 服务器已安装）

```
curl -sSL https://install.python-poetry.org | python3 -
```

验证安装

```
poetry --version
```

自动读取文件 poetry.lock 安装其中所有对应版本的依赖

```
poetry install
```

**等效命令**

| 操作 | 命令 | 等效 pip 命令 |
| --- | --- | --- |
| **安装新依赖** | poetry add <包名> | pip install <包名> |
| **安装所有依赖** | poetry install | pip install -r requirements.txt |
| **更新依赖** | poetry update # 更新全部 | pip install --upgrade <包名> |
| **删除依赖** | poetry remove <包名> | pip uninstall <包名> |
| **查看已安装依赖** | poetry show | pip list |
| **查看依赖详情** | poetry show <包名> | pip show <包名> |

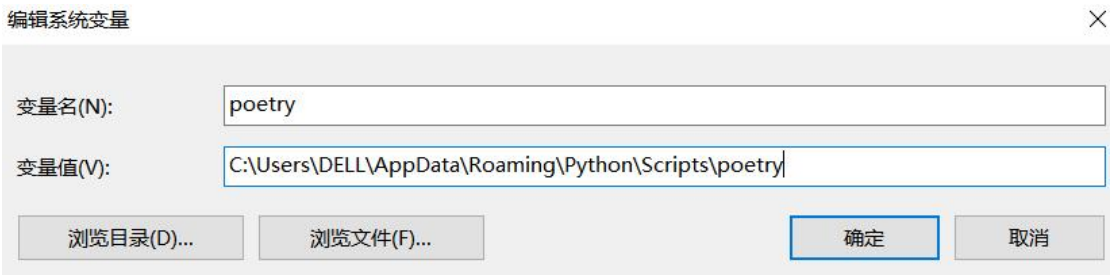
**两个文件**

文件 pyproject.toml 声明项目所需的依赖包及其版本
可以手动修改

文件 poetry.lock 记录所有依赖包的精确版本，解决环境一致性。相当于文件 requirements.txt
自动生成，禁止手动修改

**注意**

若因未添加到环境变量报错
则将路径添加到系统变量的 PATH 中
Win + R→sysdm.cpl→高级→环境变量，下例作为参考




poetry run python run.py 后修改代码不会影响运行中的进程
所以想修改代码需要先 ctrl+c 停止当前进程

poetry install 默认会安装你项目的包
如果报错则使用 poetry install --no-root 仅安装依赖

使用 Poetry 初始化一个新项目或安装依赖时，它会自动为项目创建一个虚拟环境
在项目的根目录下使用 poetry shell 就可以进入到虚拟环境（exit 退出）
虚拟环境的命名模式为“项目名-随机数-python 版本”
可以自定义存放虚拟环境的路径

```
poetry config virtualenvs.path /自定义路径
```

导出文件 requirements.txt 的命令

```
poetry export -f requirements.txt --output requirements-prod.txt --without-hashes
```

相比于 pip，Poetry 的优点：
自动分析项目中所有依赖项之间的关系，并自动解决版本冲突