

概括

使用 uv 管理 Python 版本主要有两种方式:

- 全局默认版本**: 设置 uv 默认使用哪个 Python (uv default python)
- 项目局部版本**: 在具体项目中指定使用的 Python 版本 (uv venv + pyproject.toml 或 .python-version 文件)

下面详细说明常用方法 (基于最新版 uv 0.4+, 2025年12月情况)

1. 安装并管理 Python 版本 (uv 自带 python 管理功能)

uv 本身可以直接下载和管理 Python 版本, 不需要再装 pyenv.

```
Bash
# 列出 uv 可用的 Python 版本
uv python list

# 下载并安装特定版本 (例如 3.12.7, 3.11.10 等)
uv python install 3.12.7
uv python install 3.11.10
uv python install 3.13.0 # 甚至可以使最新的 3.13

# 设置全局默认 Python 版本 (以后 uv venv 默认用这个)
uv python set-default 3.12.7
```

2. 为当前项目指定 Python 版本 (推荐方式)

方法A: 最简单, 直接在项目根目录运行 (会自动创建 .python-version 文件)

```
Bash
# 进入项目目录
cd my_project

# 指定项目使用 Python 3.12.7 (会自动下载如果没有)
uv python pin 3.12.7
```

这会产生一个 .python-version 文件, 内容就是 3.12.7, 以后在这个目录及子目录运行 uv venv 或 uv pip install 时都会自动使用这个版本.

方法B: 在 pyproject.toml 中声明 (官方推荐, 团队协作更规范)

```
toml
[project]
# 其他配置...
requires-python = ">=3.12, <3.13" # 或者精确版本 = "3.12.7"

# 或者更精确地强制使用特定 patch 版本 (uv 独有功能)
[tool.uv]
python = "3.12.7"
```

然后在项目里执行任意 uv 命令, 它都会自动使用指定的版本.

3. 手动创建虚拟环境时指定版本

```
Bash
# 明确指定 Python 版本创建虚拟环境
uv venv --python 3.12.7 .venv

# 或者使用已下载的版本别名
uv venv --python 3.12 # 会自动选最新的 3.12.x
```

4. 查看当前使用的 Python 版本

```
Bash
uv python list --only-installed # 已安装版本
uv python find 3.12            # 当前环境能找到的 3.12 版本
uv run python --version         # 当前项目实际使用的版本
```

总结最常用的一行命令 (推荐)

```
Bash
# 进入项目目录, 只需这一行就搞定版本锁定
uv python pin 3.12.7
```

以后团队成员 clone 项目后, 只需运行 uv sync 或 uv pip install -r requirements.txt 就会自动下载并使用 3.12.7, 完全不需要手动装 Python.

这样设置后, uv 的所有命令 (venv, pip, run, sync 等) 都会自动尊重你设置的 Python 版本, 非常方便.

我明明把3.11.3, 3.12.11和3.13.7安装到D:\uv\python了, 但是运行uv python list --only-installed只显示了(base) PS C:\Users\33572\Desktop> uv python list --only-installed cpython-3.13.5-windows-x86_64-none D:\anaconda3\python.exe

概括

你的问题很可能出在手动“安装”Python版本的方式上: uv 的 uv python install 命令会自动下载并管理 Python 到其默认目录 (Windows 上通常是 %USERPROFILE%\local\share\uv\python), 而你手动复制或放置文件到 D:\uv\python 后, uv 无法自动“发现”它们 (因为它们不是 uv 管理的)。结果, uv python list --only-installed 只显示了系统发现的 Anaconda Python (路径 D:\anaconda3\python.exe, 版本 3.13.5)。解决方案: 设置自定义安装目录环境变量 UV_PYTHON_INSTALL_DIR=D:\uv\python, 然后用 uv python install 重新安装版本; 同时用 --managed-python 标志过滤列表, 只显示 uv 管理的版本。

详细解释和步骤

1. 为什么只显示 Anaconda 的 Python?

- uv 在列出安装版本时, 会优先“发现”系统 Python (包括 Anaconda, 官方安装器, Windows 注册表中的 Python 等)。你的 Anaconda base 环境 (3.13.5) 被 uv 检测到并显示, 因为它在 PATH 或注册表中。
- 手动放置文件 (如你把 3.11.3, 3.12.11, 3.13.7 的 Python 放到 D:\uv\python) 不会被 uv 自动管理或发现。uv 只认识它自己通过 uv python install 下载/安装的“managed”版本。这些版本有特定目录结构 (如 cpython-3.12.11-windows-x86_64-none), 并在 Windows 注册表中注册 (符合 PEP 514)。
- only-installed 标志显示所有 uv 能“找到”的已安装版本 (系统 + managed), 但如果你没有 managed 版本, 它就只剩系统 Python 了。
- 注意: 你的提示符显示 (base), 说明你在 Anaconda base 环境中运行命令, 这可能进一步强化 uv 对 Anaconda 的偏好 (uv 会尊重当前激活的环境)。

2. 如何正确安装到自定义路径 D:\uv\python

uv 支持通过环境变量 UV_PYTHON_INSTALL_DIR 指定 managed Python 的安装目录。设置后, uv python install 会下载并安装到该路径下 (路径必须已存在, 或 uv 会创建子目录)。

步骤 (在 PowerShell 或 CMD 中执行) :

```
PowerShell
# 步骤1: 创建目录 (如果不存在)
mkdir D:\uv\python -Force

# 步骤2: 设置环境变量 (临时设置, 只影响当前会话; 想永久, 添加用户环境变量)
$env:UV_PYTHON_INSTALL_DIR = "D:\uv\python"

# 步骤3: 验证设置 (可选, 显示 uv 的存储路径)
uv python dir

# 步骤4: 重新安装你的版本 (uv 会下载到你自定义目录, 如果已存在会覆盖或用 --reinstall 强制)
uv python install 3.11.3
uv python install 3.12.11
uv python install 3.13.7 # 注意: 你之前说 3.13.7, 但列表显示 3.13.5; 用你想要的版本

# 输出示例: Installed Python 3.11.3 to: D:\uv\python\cpython-3.11.3-windows-x86_64-none
```

- 注意:
- 如果目录有非 uv 结构的文件 (如你手动放的), 建议先清空 D:\uv\python, 避免冲突。
- 下载需要网络, uv 会自动选择 Windows x86_64 架构。
- 如果想强制重新安装 (即使版本已“存在”), 加 --reinstall: uv python install 3.11.3 --reinstall。
- 永久设置环境变量: 在 Windows 设置 > 系统 > 关于 > 高级系统设置 > 环境变量 > 用户变量 > 新建 UV_PYTHON_INSTALL_DIR = D:\uv\python, 然后重启终端。

3. 如何正确列出 uv 管理的版本

默认的 uv python list --only-installed 包括系统 + managed 版本。为了只显示 uv 管理的 (忽略 Anaconda), 用 --managed-python 标志:

```
PowerShell
# 只显示 uv 管理的版本 (你的自定义目录下的)
uv python list --only-installed --managed-python

# 输出示例:
# cpython-3.11.3-windows-x86_64-none D:\uv\python\cpython-3.11.3-windows-x86_64-none\python.exe
# cpython-3.12.11-windows-x86_64-none D:\uv\python\cpython-3.12.11-windows-x86_64-none\python.exe
# cpython-3.13.7-windows-x86_64-none D:\uv\python\cpython-3.13.7-windows-x86_64-none\python.exe
```

- 其他有用标志:
- no-managed-python: 只显示系统版本 (如 Anaconda)。
- 无标志: 显示所有 (包括可下载的)。
- 如果还是只显示 Anaconda, 检查是否在 Anaconda 环境中 (运行 conda deactivate 退出 base), 或重启终端。

4. 验证和使用

验证安装:

```
PowerShell
# 检查具体版本路径
uv python find 3.12.11 # 应该返回 D:\uv\python\...\python.exe

# 测试创建 venv 用该版本
uv venv --python 3.12.11 .venv
.\venv\Scripts\Activate.ps1 # 激活
python --version # 应显示 3.12.11
```

- 如果想优先 uv managed 版本 (而非系统 Anaconda): 在 pyproject.toml 或 uv 配置中设置 python-preference = "managed" (编辑 %APPDATA%\uv\config.toml)。

5. 常见问题排查

- 还是不显示? 运行 uv python list -v (verbose) 看调试输出, 检查是否搜索到 D:\uv\python。
- Anaconda 干扰: 如果你不想用 Anaconda, 考虑用 conda deactivate 或完全卸载 base 环境; 但 uv 可以共存。
- 版本不匹配: 确保版本号精确 (如 3.12.11 而非 3.12); 用 uv python list (无 --only-installed) 查看所有可用下载版本。
- 更新 uv: 确保 uv 是最新版 (uv self update), 因为 Python 管理功能在迭代中。
- 如果问题持续, 检查 GitHub issues (如 #10049), 类似手动安装未被识别的问题常见。

这样设置后, 你的自定义目录就会被 uv 正确管理, 列表也会正常显示。❗ 还有错误输出, 贴出来我再帮调试!

我运行uv python set-default 3.11.13显示(base) PS C:\Users\33572\Desktop> uv python set-default 3.11.13 error: unrecognized subcommand 'set-default' Usage: uv.exe python [OPTIONS] <COMMAND> For more information, try '--help'.

概括

错误是因为 uv python set-default 不是有效的命令 (uv 中不存在这个子命令)。uv 不支持直接设置“全局默认 Python 版本”的命令 (如 pyenv 的 global), 而是通过项目级 pinning (使用 uv python pin 创建 .python-version 文件) 或 命令级锁定 (使用 --python 标志) 来管理版本。你的 Anaconda base 环境 (3.13.5) 会干扰 uv 的发现机制, 建议先 conda deactivate 退出 base 环境, 然后用 uv python pin 3.11.13 为当前项目 (或全局) 锁定版本。注意: 3.11.13 可能不是标准版本 (uv 支持的 CPython 版本通常是如 3.11.9 或 3.11.10; 先用 uv python list 检查可用版本)。

详细解释和步骤

1. 为什么报错?

- uv 的 python 子命令支持的子命令包括: install, list, find, pin, uninstall 等, 但没有 set-default。这是 uv 的设计: 它更注重项目隔离, 而不是系统级全局默认 (不同于 pyenv)。从 uv 官方文档 (截至 2025 年 12 月) 和 GitHub issues (如 #8135) 确认, 这个命令不存在, 用户常误用类似 pyenv 的语法。
- 你的提示符显示 (base), 表示在 Anaconda base 环境中运行, uv 会优先发现并使用系统 Python (这里是 Anaconda 的 3.13.5), 忽略 uv 管理的版本。这加剧了问题。
- 版本 3.11.13: uv 的 Python 下载列表中可能没有精确这个 patch 版本 (标准是 3.11.0 - 3.11.10 等)。运行 uv python list 查看所有可用版本, 如果没有, 会自动下载最近的兼容版 (如 3.11.10)。

2. 正确设置 Python 版本的方式

uv 优先项目级管理: 用 .python-version 文件锁定版本 (uv, pyenv 等工具都会自动读取)。全局设置用 --global 标志, 但这只是用用户级配置文件, 不是系统 PATH 修改。

步骤 (在 PowerShell 中执行, 先退出 Anaconda) :

```
PowerShell
# 步骤1: 退出 Anaconda base 环境 (避免干扰)
conda deactivate

# 步骤2: 检查可用 Python 版本 (确认 3.11.13 或最近版)
uv python list | Select-String "3.11" # 只显示 3.11 系列

# 如果现有, 安装它 (uv 会下载到你的自定义目录 D:\uv\python, 如果已设置 UV_PYTHON_INSTALL_DIR)
uv python install 3.11.13 # 如果版本不存在, 会报错; 试 3.11.10 或用 3.11 (自动选最新)

# 步骤3: 为当前项目 (或目录) 锁定版本 (推荐, 创建 .python-version 文件)
cd C:\Users\33572\Desktop\my_project # 进入你的项目目录
uv python pin 3.11.13 # 输出: Pinned '.python-version' to '3.11.13'

# 验证: 查看生成的 .python-version 文件 (内容就是 "3.11.13")
Get-Content .python-version

# 步骤4: 测试使用 (创建 venv 或 run 命令会自动用这个版本)
uv venv # 无需 --python, 它会用 pinned 版本
.\venv\Scripts\Activate.ps1
python --version # 应显示 3.11.13

# 或者直接运行脚本
uv run python --version # 自动用 pinned 版本
```

- 全局设置 (用户级默认, 影响所有无项目 pinning 的目录) :

```
PowerShell
uv python pin 3.11.13 --global # 在用户配置目录创建 .python-version
```

这会把默认锁定到 %/.config/uv/.python-version (Windows 上是 %APPDATA%\uv\.python-version)。

- 如果想临时指定 (不改文件) :

```
PowerShell
uv venv --python 3.11.13 .venv
uv run --python 3.11.13 your_script.py
```

3. 与 Anaconda 共存的注意事项

- Anaconda base 会让 uv 优先用 3.13.5, 即使 pinning 后, 如果在 base 环境中运行, uv 可能仍偏好系统版。始终 conda deactivate 后再用 uv。
- 如果想完全忽略系统 Python, 用环境变量:

```
PowerShell
$env:UV_PYTHON_PREFERENCE = "managed" # 只用 uv 管理的版本
uv python list --only-installed --managed-python # 只显示你的 D:\uv\python 中的
```

- 永久设置: 添加用户环境变量 UV_PYTHON_PREFERENCE=managed。

4. 常见问题排查

- 版本不存在: 用 uv python list 找最新版, 如 uv python pin 3.11 (自动用最新 3.11.x)。
- 还是用 Anaconda: 运行 uv python find 3.11.13 检查路径; 如果返回 Anaconda 路径, 重启终端或检查 PATH 环境变量 (Anaconda 的 Scripts 目录可能在前面)。
- 更新 uv: 确保最新版 (uv self update), 因为 Python 管理在 0.4+ 版本迭代快。
- 如果 pinning 后 venv 仍错, 用 --force: uv venv --python 3.11.13 --force。

这样设置后, uv 的所有命令 (如 uv sync, uv pip install) 都会自动用 3.11.13, 而不需每次指定。项目 clone 后, 团队只需 uv sync 即可复现环境。如果还有具体错误输出, 贴出来我继续调试!

怎么查看uv的全局python版本是否设置成功

概括

uv 没有“全局默认 Python”这个概念 (不像 pyenv 有 global), 它只有用户级默认版本 (通过 uv python pin --global 创建的 .python-version 文件)。查看是否设置成功最直接的办法就是检查该文件是否存在, 以及在没有任何项目级 .python-version 的目录下运行 uv python find 看它实际使用哪个版本。

一键验证 (推荐 3 步搞定)

```
PowerShell
# 1. 查看用户级全局 .python-version 是否存在以及内容
# Windows 路径
Get-Content "$env:APPDATA\uv\.python-version" -ErrorAction SilentlyContinue

# 或者用 uv 自己的命令查看 (uv 0.4.21+ 支持)
uv python pin --show-global

# 2. 在干净子目录 (没有任何 .python-version 文件的目录) 下查看 uv 实际会用哪个版本
# 随便新建一个空文件夹测试
mkdir C:\temp\uv-test-global
cd C:\temp\uv-test-global

uv python find # 这就是当前“全局默认”生效后的版本
# 或者更明确
uv python find --no-project

# 3. 再跑一次创建虚拟环境验证 (最直观)
uv venv .venv # 不加任何 --python 参数
.\venv\Scripts\python --version # 看这里是不是你想要的全局版本
```

正确设置全局默认版本的方法 (补全一下)

```
PowerShell
# 先确保你用的版本已经安装好
uv python install 3.11.10

# 设置用户级全局默认 (只执行一次)
uv python pin 3.11.10 --global

# 再次验证
uv python pin --show-global # 应该输出 3.11.10
```

执行完后, 任何没有自己 .python-version 文件的项目都会自动使用 3.11.10。

常见误区和额外检查方式总结

检查方式	命令	说明
查看全局文件内容	Get-Content "\$env:APPDATA\uv\.python-version"	存在即说明成功

uv 日常显示全局	uv python pin --show-global	最新版 uv 支持
当前生效版本 (无项目文件)	uv python find --no-project	最准确, 推荐

创建 venv 时实际使用的版本	uv venv → 激活后 python -V	实际效果验证
列出所有 uv 管理的版本	uv python list --only-installed --managed-python	确认版本是否真的被 uv 管理

只要第 1 步看到文件内容是你想要的版本, 并且第 2 步 uv python find --no-project 也显示相同版本, 就说明全局默认设置成功了。