pip缓存生成在哪里

概括

pip 的缓存默认存放在用户主目录下的 `~/.cache/pip` 目录（Linux/macOS）或 `%LocalAppData%\pip\Cache`（Windows）。

详细说明

pip 从 6.0 版本开始默认开启缓存机制，缓存内容主要包括：

- 已下载的轮子文件（.whl）
- 源码包（.tar.gz、.zip 等）
- HTTP 响应的 headers 等元数据

不同操作系统的默认缓存路径

操作系统	默认缓存目录	备注
Linux	`~/.cache/pip`	`~` 是用户主目录，如 `/home/username/.cache/pip`
macOS	`~/Library/Caches/pip`	同样在用户目录下
Windows	`%LocalAppData%\pip\Cache`	通常是 `C:\Users\用户名\AppData\Local\pip\Cache`
Windows（旧版）	`~\AppData\Local\pip\Cache`	早期版本可能如此

如何确认当前机器上的 pip 缓存位置

在终端/命令行直接运行下面这条命令，最准确：

```Bash
python -m pip cache dir
```

示例输出（Linux）：

```text
$ python -m pip cache dir
/home/yourname/.cache/pip
```

常见的子目录结构（进入缓存目录后）

```text
~/.cache/pip/
├── http/       # 存储 HTTP 请求的元数据缓存（v2 格式）
├── wheels/     # 存储下载过的 .whl 文件（新版默认位置）
└── selfcheck/  # pip 自检相关文件
```

如何修改或禁用缓存位置

1. 设置环境变量（临时或永久）：

```Bash
export PIP_CACHE_DIR="/自定义路径/pip-cache"
```

2. 在 pip 配置文件中永久设置（推荐）
   创建或编辑以下文件之一：
   - Linux/macOS: `~/.config/pip/pip.conf`
   - Windows: `%APPDATA%\pip\pip.ini`

   内容示例：

```ini
[global]
cache-dir = /my/custom/pip-cache
```

3. 完全禁用缓存：

```Bash
pip install xxx --no-cache-dir
```

或设置环境变量：

```Bash
export PIP_NO_CACHE_DIR=1
```

清理缓存（释放磁盘空间）

```Bash
# 查看缓存占用大小
python -m pip cache info

# 清理全部缓存
python -m pip cache purge

# 只删除超过 30 天的文件
python -m pip cache remove "*/wheels/*" --days 30
```

总结：99% 的情况下，直接运行 `python -m pip cache dir` 就能立刻知道你当前 pip 的缓存到底生成在哪里。

怎么改变缓存路径

概括

改变 pip 缓存路径有 3 种常用方式（推荐度从高到低）：

1. **推荐：修改 pip 配置文件**（永久生效，全局最优雅）
2. 设置环境变量 `PIP_CACHE_DIR`（简单，适合临时或脚本使用）
3. 每次命令加 `--cache-dir` 参数（最灵活，但每次都要写）

下面详细介绍每种方法的具体操作步骤。

方法 1：修改 pip 配置文件（永久修改，全局生效，强烈推荐）

Linux / macOS

1. 创建配置文件目录（如果不存在）：

```bash
mkdir -p ~/.config/pip
```

2. 创建或编辑配置文件：

```bash
nano ~/.config/pip/pip.conf
```

3. 写入以下内容（把路径改成你想要的）：

```ini
[global]
cache-dir = /你的自定义路径/pip-cache
```

示例：

```ini
[global]
cache-dir = /data/pip-cache
# 或者放到大容量硬盘分区
cache-dir = /mnt/ssd/pip-cache
```

Windows

1. 打开资源管理器，输入路径并回车：

```text
%APPDATA%
```

2. 创建文件夹 `pip`（如果没有）
3. 在里面新建文本文件，命名为 `pip.ini`
4. 编辑内容：

```ini
[global]
cache-dir = D:\pip-cache
```

（路径支持中文和空格，但建议用英文路径）

保存后立即生效，下次 pip 下载就会使用新路径。

方法 2：设置环境变量（快速临时修改）

Linux / macOS（临时，关闭终端失效）

```bash
export PIP_CACHE_DIR="/你的新路径/pip-cache"
pip install requests   # 就会用新路径
```

Linux / macOS（永久生效）

编辑 `~/.bashrc` 或 `~/.zshrc`，加入一行：

```bash
export PIP_CACHE_DIR="/data/pip-cache"
```

Windows（永久生效）

1. 右键「此电脑」→ 属性 → 高级系统设置 → 环境变量
2. 在「用户变量」里新建：
   变量名：`PIP_CACHE_DIR`
   变量值：`D:\pip-cache`

方法 3：每次命令都加参数（最灵活）

```bash
pip install django --cache-dir /tmp/my-pip-cache
pip install numpy --cache-dir D:\temp-pip-cache
```

验证是否成功修改

运行下面这条命令，无论用哪种方式都能立刻看到当前真实缓存路径：

```bash
python -m pip cache dir
```

如果输出是你设置的新路径，就说明成功了！

小技巧：把缓存目录做成软链接（兼容性最好）

如果你不想改配置，又想“假装”改路径，可以用软链接（符号链接）：

```bash
# Linux/macOS
sudo rm -rf ~/.cache/pip                  # 先删除旧的（小心！）
ln -s /你的新路径/pip-cache ~/.cache/pip   # 建立软链接

# Windows（用管理员 PowerShell）
rmdir "$env:LOCALAPPDATA\pip\cache"
New-Item -ItemType SymbolicLink -Path "$env:LOCALAPPDATA\pip\cache" -Target "D:\pip-cache"
```

这样所有程序（包括旧版本 pip）都自动认为缓存还在默认位置，但实际数据存在你指定的大磁盘上。

总结一句话：
最推荐做法 = 创建或修改 pip 配置文件（pip.conf 或 pip.ini），加入 cache-dir 一行，一次设置永久生效。

1.1秒 快速模式

pip缓存有什么用

概括

pip 缓存的主要作用是**大幅加速包的重复安装和离线安装**，同时**节省网络流量和时间。**
简单说：第一次下载慢，之后所有操作（包括在其他虚拟环境、其他机器、CI/CD）几乎都是秒装。

pip 缓存到底能帮你省多少时间？（真实对比）

操作场景	不开缓存（--no-cache-dir）	开缓存（默认）	加速倍数
第二次在同一个虚拟环境安装	重新下载（几秒～几分钟）	直接本地复制（<1秒）	10～300倍
创建全新虚拟环境，装相同依赖	重新下载	直接复制缓存	10～100倍
公司/学校网络很慢或限流量	每次都要完整下载	只下载一次	节省 90%+ 流量
CI/CD（GitHub Actions 等）	每次构建都重新下载	加 cache 步骤秒级	构建时间减半
离线电脑安装包	完全无法安装	复制缓存即可安装	从不可能→可能

pip 缓存具体能缓存哪些内容（非常实用）

缓存内容	具体用途	典型例子
.whl 二进制轮子文件	安装速度最快，无需编译	numpy、pandas、scipy 等大包
源码包（.tar.gz）	即使没有轮子，也缓存源码，第二次不需要再下载	一些冷门包或旧版本
HTTP headers 元数据	快速判断是否有新版本，不用每次都完整请求 PyPI	加速 pip install -U 检查
多版本、多平台轮子	同时支持 manylinux、macos、windows、arm、python37~python312 等所有组合	不同机器之间可以直接共享缓存

实际生产/开发中常见的超实用场景

1. **团队共享缓存（省几百 GB 流量）**
   把缓存目录挂载到 NAS 或公司内部镜像源，所有人共用一个缓存。
2. **离线空气隔离环境安装 Python 包**
   在有网机器执行：

   ```bash
   pip download pandas numpy -d /tmp/offline-cache
   ```

   把整个缓存目录拷贝到无网机器，直接离线安装：

   ```bash
   pip install pandas numpy --no-index --find-links /path/to/offline-cache
   ```

3. **GitHub Actions 加速构建（官方推荐）**
   加入缓存步骤后，构建时间从 3～5 分钟 → 30 秒：

   ```yaml
   - uses: actions/cache@v4
     with:
       path: ~/.cache/pip
       key: pip-$$ {{ runner.os }}- $${{ hashFiles('requirements.txt') }}
   ```

4. **Docker 镜像瘦身 + 构建加速**
   在 Dockerfile 中提前 COPY 缓存，或者用 --cache-dir 挂载主机缓存。

什么时候可以安全清理缓存?

- 磁盘空间不够了 → 直接 `pip cache purge`（几 GB 很常见）
- 想强制重新下载最新版 → 加 `--no-cache-dir`
- 缓存损坏了（极少见）→ 删除整个缓存目录，pip 会自动重建

一句话总结

pip 缓存 = Python 生态的“本地软件源”
第一次慢一点，之后所有机器、所有虚拟环境、所有构建流程都能瞬间完成安装，是真正能省时间、省流量、省磁盘磨损的“隐形加速器”。
建议：永远不要禁用缓存，除非你有特殊理由。

1.1秒 快速模式

pip缓存清理技巧

conda缓存管理

让 Think 更努力思考

How can Grok help?

自动模式