Managing Third-Party Dependencies With pip

1. **Welcome & Course Overview** ✅
2. **Managing Third-Party Dependencies With pip**
3. Isolating Dependencies With Virtual Environments
4. Finding Quality Python Packages
5. Setting Up Reproducible Environments & Application Deploys
6. Course Conclusion

Managing Third-Party Dependencies With pip

1. Introduction to Dependency Management
2. pip: The Python Package Manager
3. Installing & Updating pip
4. Python Package Repositories
5. Installing Packagages With pip
6. Identifying & Updating Outdated Packages
7. Uninstalling Packages

What is Dependency Management and what do you need it for?

Dependencies ≔

Most "real world" Python programs use 3rd-party libraries & frameworks. They're called *dependencies*.

Example:

```
my_program
│
├── super_library
│
└── other_dependency
```

Transitive Dependencies :=

"Dependencies of Dependencies", "secondary dependencies"

```
my_program
│
├── super_library
│   ├── decent_library
│   └── tool_v2.1
│       └── awesome_framework
│
└── other_dependency
    └── mega_module
```

Managing dependencies **manually** is difficult:

- time-consuming
- copying/pasting source code makes updates tedious
- easy to make mistakes
- …

The Solution: Package Managers

pip—The Python Package Manager

What are "packages" in Python?

Package ≔

A bundle of software to be installed into a Python environment. Typically 3rd-party libraries and frameworks.

Examples:

- Django
- Requests
- NumPy

pip—the Python Package Manager

pip comes with any modern Python install

Its main interface is a command line tool:

```
$ pip ...
```

Demo

Installing & Updating pip

pip should be installed on Python 2.7.9+ and Python 3.4+

What if it isn't?

Installing pip

Option 1: Upgrade to a more modern Python

Installing pip

Option 2: Add pip to your existing Python install

macOS & Windows:

- Download https://bootstrap.pypa.io/get-pip.py
- `$ python get-pip.py`

Linux (Debian/Ubuntu):

- `$ sudo apt update`
- `$ sudo apt install python-pip`

packaging.python.org/installing/

Updating pip to the latest version

macOS:

```
$ sudo pip3 install --upgrade pip setuptools
```

Linux (Debian/Ubuntu):

```
$ sudo apt update && sudo apt upgrade python-pip
```

Windows:

```
C:\>pip install --upgrade pip setuptools
```

Demo

Python Package Repositories

Python packages are collected in **software repositories**:

- The biggest one/official one is called **PyPI**
  (or the "cheese shop")

PyPI

Developers can register for a (free) PyPI account and submit new packages to the repository

Once a package appears in PyPI, everyone else can install it through pip

(There's no review or QA process)

Searching for packages on PyPI

PyPI website → pypi.python.org

Demo

Searching for packages from the command line

```
$ pip search [name]
```

PyPI vs Warehouse

PyPI vs Warehouse

PyPI is getting superseded by **Warehouse**

Warehouse is currently in beta testing

(Don't worry about the transition)

Warehouse Sneek Peek

Warehouse (beta) → pypi.org

Demo

Installing Packages With pip

Demo

Installing packages with pip

```
$ pip install [name]
```

- Installs package from PyPI
- Packages are cached locally to speed up repeated installs

Package version specifiers

```
$ pip install requests==2.1.3
```

Package version specifiers

```
$ pip install requests==2.1.3

$ pip install requests>=2,<3
```

Package version specifiers

```
$ pip install requests==2.1.3

$ pip install requests>=2,<3

$ pip install requests~=2.1.3
```

→ any 2.1.X version >= 2.1.3

Warning: Installing packages globally

⚠️ Using pip this way **installs packages into the global environment**

- Okay if done intentionally (e.g. for Python-based command-line tools like `httpie`)

⚠️ Using pip this way **installs packages into the global environment**

- Okay if done intentionally (e.g. for Python-based command-line tools like `httpie`)
- Most of the time you should prefer **virtual environments**
  - They keep Python packages nice and separate by project

Installing packages from Git (and other sources)

```
$ pip install git+https://github.com/user/repo.git@branch
```

Installing packages from Git (and other sources)

```
$ pip install git+https://github.com/user/repo.git@branch
```

Examples:

```
# Install from branch:
$ pip install git+https://github.com/kennethreitz/requests.git@master
```

Installing packages from Git (and other sources)

```
$ pip install git+https://github.com/user/repo.git@branch
```

Examples:

```
# Install from branch:
$ pip install git+https://github.com/kennethreitz/requests.git@master

# Install from commit hash:
$ pip install git+https://github.com/kennethreitz/requests.git@2aaf6ac
```

Installing packages from Git (and other sources)

```
$ pip install git+https://github.com/user/repo.git@branch
```

Examples:

```
# Install from branch:
$ pip install git+https://github.com/kennethreitz/requests.git@master

# Install from commit hash:
$ pip install git+https://github.com/kennethreitz/requests.git@2aaf6ac

# Install from tag/release:
$ pip install git+https://github.com/kennethreitz/requests.git@v2.13.0
```

Identifying & Updating Outdated Packages

Identifying & Updating Outdated Packages

```
$ pip list --outdated

$ pip install --upgrade [name]
```

Demo

Uninstalling packages

Uninstalling packages

```
$ pip uninstall [name]
```

⚠️ Uninstalling **secondary dependencies**?

⚠️ Uninstalling secondary dependencies?

- `pip uninstall` **does not** uninstall secondary dependencies
- Yet another reason to use **virtual environments**:
  → Delete & re-create them to remove unneeded secondary deps

Demo

Managing Third-Party Dependencies With pip ✅

1. **Introduction to Dependency Management** ✅
2. **pip: The Python Package Manager** ✅
3. **Installing & Updating pip** ✅
4. **Python Package Repositories** ✅
5. **Installing Packagages With pip** ✅
6. **Identifying & Updating Outdated Packages** ✅
7. **Uninstalling Packages** ✅

Summary

- **Dependency management** enables modern software development by making well-packaged building blocks available for use in your own programs.
- Key tool: **pip**—Python's recommended **package manager**
- Python packages are hosted on package repositories (**PyPI**)
- pip has powerful **version management features**