Менеджмент версий Python

Ручной способ

Установка из стандартного репозитория пакетного менеджера >

В различных сборках Linux можно установить python через соответствующий пакетный менеджер, но часто бывает такое, что они могут конфликтовать друг с другом или просто отсутствовать в репозиториях нужная версия Python.

```
root@domain-server:~$ apt install python3.8
Reading package lists... Done
Building dependency tree... Done
Package python3.8 is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source
E: Package 'python3.8' has no installation candidate
```

Рассмотрим установку Python из исходников вручную:

```
# -1. Обновление репозиториев
sudo apt update

# 0. Установка зависимостей для сборки
sudo apt install -y make build-essential libssl-dev zlib1g-dev \
        libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm \
        libncurses5-dev libncursesw5-dev xz-utils tk-dev libffi-dev \
        liblzma-dev python3-openssl

# 1. Скачивание исходного кода Python
wget https://www.python.org/ftp/python/3.8.4/Python-3.8.4.tgz

# 2. Распаковка
tar xvf Python-3.8.4.tgz && cd Python-3.8.4

# 3. Конфигурация сборки: добавляем pip и оптимизации
./configure --enable-optimizations --with-ensurepip=install

# 4. Сборка python с указанием количества используемых потоков
make -j 4

# 5. Установка (если хотим сделать доступным в системе глобально)
sudo make altinstall
```

```
# 6. Проверка
python3.8 --version
```

Плюсы:

- Контролируете все этапы;
- Точечно можете настроить все параметры сборки и установки;
- Используется нативно, как системный Python;

Минусы:

- Нужно собирать руками, каждый раз отдельные версии;
- Бывают проблемы с одновременной установкой нескольких одинаковых мажорных версий (`python3.8` может вызвать как `python3.8.2`, так и `python3.8.16`);

pyenv

> Это как если бы мы хотели автоматизировать ручную сборку и добавить несколько удобных вещей из коробки.

Pyenv это инструмент для управления различными версиями Python с собственной экосистемой плагинов.

Установка

```
# -1. Обновление репозиториев
sudo apt update

# 0. Установка зависимостей для сборки
sudo apt install -y make build-essential libssl-dev zlib1g-dev \
        libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm \
        libncurses5-dev libncursesw5-dev xz-utils tk-dev libffi-dev \
        liblzma-dev python3-openssl

# 1. Установка pyenv
curl https://pyenv.run | bash

# 2. Добавление в shell (.bashrc, .zshrc etc.)
echo -e 'export PYENV_ROOT="$HOME/.pyenv"' >> .bashrc
echo -e 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >>
.bashrc
echo -e 'eval "$(pyenv init -)"' >> .bashrc
echo -e 'eval "$(pyenv virtualenv-init -)"' >> .bashrc

# 3. Перезагружаем shell
exec "$SHELL"

# 4. Проверяем
```

```
pyenv --version
```

Пример использования для нескольких проектов разных версий Python

```
pyenv install --list # смотрим что у нас есть

# Устанавливаем версии, которые хотим
pyenv install 3.6.15 3.11.3

# Отображает список всех версий Python
pyenv versions

# Создаем venv конкретной версии
pyenv virtualenv 3.6.15 someproject-venv

# Создаем папку проекта
mkdir someproject && cd someproject

# Связываем папку проекта с нужной версией venv
pyenv local someproject-venv

# Проверяем
python -V
which python
pyenv which python # Показывает настоящее расположение venv

# Пример с другим проектом и другой версией Python
mkdir ../otherproject && cd ../otherproject
pyenv virtualenv 3.11.3 otherproject-venv
pyenv local otherproject-venv
python -V
which python
pyenv which python
cat .python-version # определяет какой venv активировать внутри директории

pyenv global 3.6.15 # Делает доступным глобально нужную версию python.
                                        # Системный python по умолчанию
называется `system`

pyenv shell 3.11.3 # Определяет в сессии терминала конкретную версию
```

Плюсы:

- Легко управлять десятками версий Python и создавать виртуальные окружения для них;
- Позволяет работать не только с CPython;

- Автоматизирует рутинные действия по работе с виртуальными окружениями;
- Подключается по абсолютным путям в IDE, как и обычный venv;

Минусы:

- Работает только с Python;
- При установке по умолчанию нужно помнить, что pyenv использует не всегда оптимальные параметры сборки. Можно передавать дополнительные флаги оптимизации кода и установки дополнительных модулей Python через env переменные, но время сборки может значительно увеличиться.

Docker

Самый простой способ получить окружение Python при наличии установленного Docker - взять образ с нужной версией Python

> **Совет**
>
> Все поддерживаемые на данный момент версии Python можно найти в официальном Docker Hub Python

Запустить терминал в нужном образе можно командой:

```
docker run -it python:3.9.16 bash
```

Также можно собрать собственный образ с нужной версией Python из исходников, чтобы таким образом создать себе полностью выделенное окружение для разработки без вмешательства в хост систему.

Пример

```
FROM almalinux:8.6
RUN dnf update -y && dnf install wget yum-utils \
        make gcc openssl-devel bzip2-devel libfi-devel zlib-devel sqlite-
devel -y
RUN wget https://www.python.org/ftp/python/3.10.2/Python-3.10.2.tgz \
        && tar xzf Python-3.10.2.tgz \
        && rm Python-3.10.2.tgz
WORKDIR /Python-3.10.2
RUN ./configure --with-system-ffi --with-computed-gotos --enable-loadable-
sqlite-extensions \
        && make \
        && make altinstall
RUN python3.10 -V
```

Вы можете поискать существующие мультиверсионные сборки Python образов. Они бывают удобны для специфических задач

С помощью соверменных IDE и редакторов кода такой запущенный контейнер можно подключить как виртуальную среду для проекта или вести разработку непосредственно в контейнере.

Плюсы:

- все лежит в контейнере: легко поставить, легко удалить, если стал не нужен;
- Легко использовать для нескольких проектов: из одного образа можно создать необходимое число контейнеров (или своих образов), которые будут занимать минимум места при правильной сборке;

Минусы:

- Нужен Docker;
- Нужно уметь хорошо пользоваться Docker, чтобы не раздувались контейнеры и работать было комфортно;
- Инструменты разработки должны уметь подключаться к такой конфигурации (PyCharm CE не умеет, vim/nvim тоже, но их можно установить напрямую)
- Больше нагрузка на компьютер

rye

Появился всего пару недель назад в open source на Github. Проект от создателя Flask Армина Ронахера.

(Не путайте с библиотекой автоматизации)

Вот тут он отвечает на вопрос зачем он его сделал.

Работает на основе переносимых версий Python.

Не дает возможности использовать `pip` в виртуальном окружении вообще.

```
# -1. Нужно установить Rust в частности нам нужен cargo для сборки rye
curl https://sh.rustup.rs -sSf | sh
# После установки возможно потребуется перезагрузить терминал: exec "$SHELL"

# 0. Сборка проекта
cargo install --git https://github.com/mitsuhiko/rye rye

# 1. Инициализация проекта
rye init my_best_project && cd my_best_project

# 2. Выбираем нужную версию Python
rye pin cpython@3.11

# 3. Добавляем зависимости
```

```
rye add flask  # or rye add "flask>=2.0"
rye add --dev black

# 4. Устанавливаем
rye sync

# 5. Что-то делаем в проекте
...

# 6. Запускаем
rye run black
```

Полезные команды

```
# показать статус проекта
rye show

# установить пакет глобально НО
# чтобы он был доступен необходимо добавить ~/.rye/shims в PATH
rye install <something>

# Установка не из PIP
rye install 'datasette @ https://github.com/simonw/datasette/archive/main.zip'

# Просмотр всех доступных скриптов
rye run

# Список установленных версий python
rye toolchain list
```

Пример pyproject.toml из rye проекта >

```
[project]
name = "my-best-project"
version = "0.1.0"
description = "Add a short description here"
dependencies = ["flask~=2.3.2", "gunicorn~=20.1.0"]
readme = "README.md"
requires-python = ">= 3.8"

[build-system]
requires = ["hatchling"]
build-backend = ["hatchling.build"]

[tool.rye]
managed = true
dev-dependencies = ["black~=23.3.0"]
```

```
[tool.rye.scripts]
flask_server = "flask --app my_best_project.main run --host 0.0.0.0"
python_server = "python -m http.server 8000"
```

Плюсы:

- Не нужно ничего компилировать
- Умеет устанавливать программы в изолированные глобальные окружения как pipx
- Умеет почти тоже что и poetry, и pyenv, но с ограничениями

Минусы:

- Нет полноценной документации
- Доступных версий Python значительно меньше чем в pyenv
- rye не работает под Windows
- Нужно устанавливать Rust и собирать rye самому

Полезные ссылки

pyenv

1. Pyenv Source Code
2. Pyenv Wiki
3. Intro to pyenv - Real Python
4. Современный Python: как начать свой проект с нуля при помощи Pyenv и Poetry - Хабр
5. Pyenv Win

Остальное

1. Python 3 Installation & Setup Guide - Real Python
2. Multi python container - Docker Hub (! archived !)
3. Работа с Python в контейнере VS Code
4. Работа с Python в контейнере PyCharm Professional
5. Управление зависимостями с помощью Conda
6. Rye