Системы виртуализации проектов на Python

Virtualenv

Перед тем, как начать: если вы не пользуетесь Python 3, вам нужно будет установить **инструмент virtualenv** при помощи pip:

```
$ pip install virtualenv
```

Создание новой виртуальной среды внутри каталога:

```Shell
# Python 2:
$ virtualenv env

# Python 3
$ python3 -m venv env
```

Эта команда создает каталог под названием «*env*», структура каталога которого схожа со следующей:

```
├── bin
│   ├── activate
│   ├── activate.csh
│   ├── activate.fish
│   ├── easy_install
│   ├── easy_install-3.5
│   ├── pip
│   ├── pip3
│   ├── pip3.5
│   ├── python -> python3.5
│   ├── python3 -> python3.5
│   └── python3.5 -> /Library/Frameworks/Python.framework/Versions/3.5/bin/pyt
├── include
├── lib
│   └── python3.5
│       └── site-packages
└── pyvenv.cfg
```

Краткое описание содержимого папок:

- **bin** – файлы, которые взаимодействуют с виртуальной средой;
- **include** – C-заголовки, компилирующие пакеты Python;
- **lib** – копия версии Python вместе с папкой «*site-packages*», в которой установлена каждая зависимость.

Чтобы использовать пакеты (или ресурсы) среды в изоляции, нужно «*активировать*» их.

```Shell
source env/bin/activate
```

Выйти из виртуального окружения можно командой:

```
(env) $ deactivate
```

Venv

Основное отличие venv в том, что он встроен в интерпретатор и может отрабатывать ещё до загрузки системных модулей. Для этого, при определении базовой директории с библиотеками, используется примерно такой алгоритм:

- ✓ в директории с интерпретатором или уровнем выше ищется файл с именем pyvenv.cfg;
- ✓ если файл найден, в нём ищется ключ home, значение которого и будет базовой директорией;
- ✓ в базовой директории идёт поиск системной библиотеки (по спец. маркеру os.py);
- ✓ если что-то пошло не так – всё откатывается к захардкоженному в бинарнике значению.

Чтобы создать окружение, нужно вызвать через ключ -m модуль venv, либо использовать встроенный скрипт pyvenv:

pyvenv /path/to/new/venv

Скрипт создаст указанную директорию, вместе со всеми родительскими директориями, если потребуется, и построит виртуальное окружение. Это можно делать и в Windows:

c:\Python33\python -m venv /path/to/new/venv

При создании можно добавлять различные параметры, как, например, включение системных site-packages или использование symlink вместо копирования интерпретатора.

В отличии от virtualenv новый venv требует чтобы создаваемая директория не существовала, либо была пустой.

Есть два метода активации окружения venv:

- ✓ метод активации через bin/activate (Scripts/activate в windows):

  ```
  cd /path/to/new/venv

  .bin/activate

  python3 some_script.py
  ```

- ✓ лишь вызвать интерпретатор из окружения и всё сработает автоматически:

  /path/to/new/venv/bin/python3 some_script.py
  Если в системе обновилась версия python, то виртуальное окружение иногда тоже нужно обновить. Всё просто – вызываем venv аналогично созданию окружения, добавив ключ --upgrade:

  pyvenv --upgrade /path/to/new/venv

Деактивация виртуального окружения
Деактивация выполняется командой *deactivate* (работает как в *Windows,* так и в *Linux*).

Pipenv

Pipenv - это пакетный инструмент для Python, который решает некоторые распространенные проблемы, связанные с типичным рабочим процессом. В дополнение к решению некоторых распространенных проблем, он объединяет и упрощает процесс разработки с помощью единого инструмента командной строки.

Для инициализации виртуальной среды Python 3, выполните команду:

pipenv shell

Если виртуальной среды еще не существует, она будет создана. Также вы можете принудительно создать среду Python 2 или Python 2. Для этого нужно выполнить команды

pipenv --three или pipenv –two

Устанавливаем пакеты

Далее можно установить сторонний пакет. Существует 3 варианта установки:

Вариант #1. Указываем пакет, который нужно установить

pipenv install scrapy

Установится последняя версия пакета. При этом в Pipfile добавится запись scrapy = "*"

Вариант #2. Не указываем пакет (установка пакетов из Pipfile)

pipenv install

Установятся все пакеты из Pipfile и обновится Pipfile.lock

Вариант #3. Не указываем пакет (установка пакетов из Pipfile.lock)

pipenv sync

Установятся все пакеты из Pipfile.lock

Запускаем скрипты

Запустить python-скрипты можно несколькими способами:

Способ #1. Активируем виртуальное окружение командой pipenv shell, далее выполняем python script-name.py

Способ #2. Запустить скрипт сразу внутри оболочки virtualenv можно командой:

pipenv run python script-name.py

Pyenv

Pyenv можно установить, используя автоматический скрипт.

После этого появится сообщение о том, что необходимо добавить следующие строки кода в .profile / .bash_profile для того, чтобы автоматически обнаруживать pyenv.

```
export PATH="$HOME/.pyenv/bin:$PATH"
eval "$(pyenv init -)"
eval "$(pyenv virtualenv-init -)"
```

Инструмент располагается в ~/.pyenv/, а все версии будущих интерпретаторов Python будут находиться ~/.pyenv/versions/.

Список команд pyenv можно увидеть следующим образом:

$ pyenv commands

Справка по каждой команде:

$ pyenv local –help

Конфигурационные файлы

Пример конфигурационного файла requirements.txt (Venv,virtualenv, etc.)

```
user@arb:/usr/home/test2# cat ENV/pyvenv.cfg
home = /usr/bin
include-system-site-packages = false
version = 3.7.0
user@arb:/usr/home/test2# readlink ENV/bin/python3
/usr/bin/python3
```

Пример конфигурационного файла Pipfile (pipfile, pyenv, etc.)

```
[[source]]
url = 'https://pypi.python.org/simple'
verify_ssl = true
name = 'pypi'

[requires]
python_version = '2.7'

[packages]
requests = { extras = ['socks'] }
records = '>0.5.0'
django = { git = 'https://github.com/django/django.git'
"e682b37" = {file = "https://github.com/divio/django-cm
"e1839a8" = {path = ".", editable = true}
pywinusb = { version = "*", os_name = "=='nt'", index="

[dev-packages]
nose = '*'
unittest2 = {version = ">=1.0,<3.0", markers="python_ve
```

Requirements.txt	pipfile
Обычный текстовый документ с путями и названиями пакетов python, их версия и условие, больше, меньше, равно	Синтаксис TOML — позволяет определять более точную конфигурацию. Группы зависимостей (одна по умолчанию, другая для разработки, третья для дистрибутивов и т. д.), поэтому больше не нужно распределять зависимости по разным файлам.
	Pipfile.lock — специальный файл (в дополнение к pipfile), который связывает версии пакетов и дополнительно обеспечивает некоторую безопасность, используя для каждого пакета хэш-код (Pipenv)
	Группы зависимостей (одна по умолчанию, другая для разработки, третья для дистрибутивов и т. д.), поэтому больше не нужно распределять зависимости по разным файлам.