|  |  |
| --- | --- |
|  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных Технологий

Кафедра Корпоративных Информационных Систем

**ОТЧЕТ**

по Практической Работе №1

на тему

«Менеджеры пакетов»

по дисциплине

«Конфигурационное управление»

Выполнил студент группы ИКБО-11-19 Коннов А.Д

Принял Жигалов К. Ю.

Выполнено «30» сентября 2020 г.

Зачтено «\_\_\_» \_\_\_\_\_\_\_\_\_ 2020 г.

Москва, 2020

# Цель работы

Освоение понятия менеджера пакетов.

# Постановка задачи

1. Разобраться, что собой представляют менеджеры пакетов в реализациях популярных языков программирования (Java, JavaScript, Python и так далее). Как устроен пакет.
2. Описать в псевдокоде алгоритм работы выбранного менеджера пакетов.
3. Реализовать на выбранном ЯП загрузку пакета по его имени, без использования возможностей существующего менеджера пакетов. Реализовать вывод метаданных пакета.
4. Разработать программу, которая по названию пакета строит граф его зависимостей. Изобразить граф в формате Graphviz.

# Решение

* 1. Менеджер пакетов.

Менеджер пакетов следит за тем, какие программы установлены на вашем компьютере, и позволяет вам легко устанавливать новые программы, обновлять программы до более новых версий или удалять те программы, которые вы ранее установили. Как предполагает само название, менеджеры пакетов имеют дело с пакетами — наборами файлов, которые упакованы вместе и могут быть установлены или удалены группой.

У каждого пакетного менеджера есть файл с настройками, в котором нам нужно указать от каких пакетов зависит наш код, чтобы пакетный менеджер их скачал и установил к нам в систему. При этом каждый пакет может зависеть от других пакетов. Пакетный менеджер распутывает эту систему зависимостей и устанавливает всё что нужно, поэтому их ещё называют менеджерами зависимостей.

* 1. Загрузка пакета.

Django также является обычным пакетом Python. Приложение Django — это пакет Python, который специально предназначен для использования в проекте Django. Приложение может использовать общие соглашения Django.

Механизм работы, в принципе, не сильно отличается от его аналогов в других средах: есть центральный репозиторий, к которому обращается пользователь и откуда получают информацию о том, что и откуда загружать. Существенный недостаток заключается в том, что формат пакетов (по сути — zip-архивов) никак не стандартизирован.

Пакет упрощает установку и обновление зависимостей проекта, то есть сторонних библиотек, которые он использует.

* 1. Метаданные пакета.

Алгоритм загружает пакет по его введённому пользователем имени, при этом не происходит использование уже существующего менеджера пакетов.



Рис.1 Вывод метаданных на экран

* 1. Граф зависимостей.

Полученный граф зависимости пакетов на примере пакета Django:

digraph G {

"django" -> "pytz"

"django" -> "asgiref"

"django" -> "sqlparse"

}

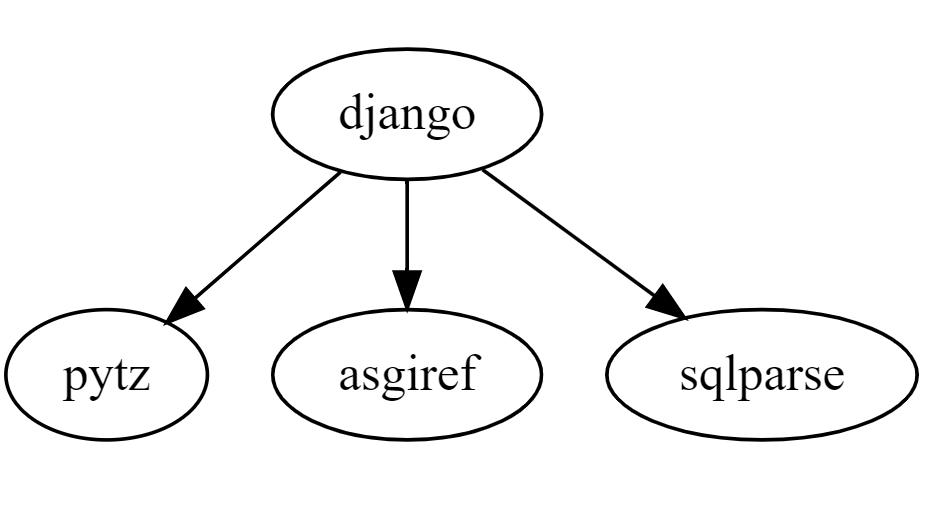


Рис.2 Визуализация построенного графа в формате Graphviz

# Вывод

В результате выполнения работы я:

1. Ознакомился с теорией менеджеров пакетов
2. Узнал алгоритм работы менеджеров пакетов
3. Научился интерпретировать реализацию загрузку пакета по его имени, а также реализацию вывода графа зависимостей менеджера пакетов
4. **Код программы**

import urllib.request

import xml.etree.ElementTree as ET

import zipfile

import io

def load(url):

with urllib.request.urlopen(url) as f:

data = f.read()

return data

def get\_package\_url(name):

data = load("https://pypi.org/simple/%s/" % name)

root = ET.fromstring(data)

package\_url = None

for elem in root[1]:

if elem.tag == "a":

url = elem.attrib["href"]

if ".whl#" in url:

package\_url = url

return package\_url

def get\_package\_dependencies(url):

data = load(url)

obj = io.BytesIO(data)

zipf = zipfile.ZipFile(obj)

meta\_path = [s for s in zipf.namelist() if "METADATA" in s][0]

with zipf.open(meta\_path) as f:

meta = f.read().decode("utf-8")

print(meta)

deps = []

for line in meta.split("\n"):

line = line.replace(";", " ").split()

if not line:

break

if line[0] == "Requires-Dist:" and "extra" not in line:

deps.append(line[1])

return deps

def get\_pypi\_graph(name):

graph = {}

def rec(name):

print(name)

graph[name] = set()

url = get\_package\_url(name)

if not url:

return

deps = get\_package\_dependencies(url)

for d in deps:

graph[name].add(d)

if d not in graph:

rec(d)

rec(name)

return graph

def gv(graph):

lines = ["digraph G{"]

for v1 in graph:

for v2 in graph[v1]:

lines.append('"%s" -> "%s"' % (v1,v2))

lines.append("}")

return "\n".join(lines)

g = get\_pypi\_graph("django")

print(gv(g))