

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Установка пакетов в Python. Виртуальные окружения»**

**Отчет по лабораторной работе № 2.14
по дисциплине «Основы программной инженерии»**

Выполнил студент группы

ПИЖ-б-о-21-1

Трушева В. О. .« » 2023г.

Подпись студента _____

Работа защищена « _____ 20__ г.

Проверила Воронкин Р.А. _____

(подпись)

Ставрополь 2022

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

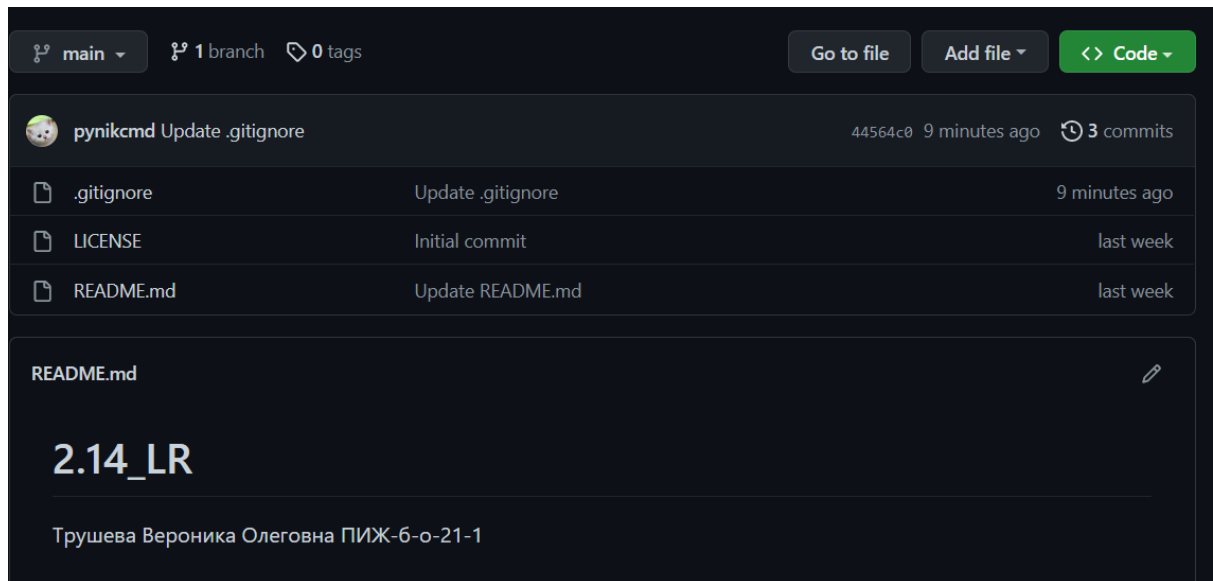


Рисунок 1 – Созданный репозиторий

3. Выполните клонирование созданного репозитория.
4. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.

```
D:\fgit\2.14_LR_OPI>git flow init

Which branch should be used for bringing forth production releases?
- main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [D:/fgit/2.14_LR_OPI/.git/hooks]
```

Рисунок 2 – Модель ветвление git flow

5. Создайте виртуальное окружение Anaconda с именем репозитория.

```
Anaconda Prompt (Anaconda3)

(base) C:\Users\Admin>conda create -n 2.14lr
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 22.9.0
  latest version: 23.1.0

Please update conda by running

    $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\Admin\.conda\envs\2.14lr

Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate 2.14lr
#
# To deactivate an active environment, use
#
#     $ conda deactivate

Retrieving notices: ...working... done

(base) C:\Users\Admin>conda activate 2.14lr

(2.14lr) C:\Users\Admin>
```

Рисунок 3 – Виртуальное окружение Anaconda

6. Установите в виртуальное окружение следующие пакеты: `pip`, `NumPy`, `Pandas`, `SciPy`.

```
(2.14lr) C:\Users\Admin>conda install -n 2.14lr pip
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
current version: 22.9.0
latest version: 23.1.0
```

Please update conda by running

```
$ conda update -n base -c defaults conda
```

```
## Package Plan ##
```

```
environment location: C:\Users\Admin\.conda\envs\2.14lr
```

```
added / updated specs:
- pip
```

The following packages will be downloaded:

package	build	
certifi-2022.12.7	py310haa95532_0	149 KB
openssl-1.1.1t	h2bbff1b_0	5.5 MB
pip-22.3.1	py310haa95532_0	2.8 MB
python-3.10.9	h966fe2a_0	15.8 MB
setuptools-65.6.3	py310haa95532_0	1.2 MB
wincertstore-0.2	py310haa95532_2	15 KB
Total:		25.4 MB

The following NEW packages will be INSTALLED:

bzip2	pkgs/main/win-64::bzip2-1.0.8-he774522_0	None
ca-certificates	pkgs/main/win-64::ca-certificates-2023.01.10-haa95532_0	None
certifi	pkgs/main/win-64::certifi-2022.12.7-py310haa95532_0	None
libffi	pkgs/main/win-64::libffi-3.4.2-hd77b12b_6	None
openssl	pkgs/main/win-64::openssl-1.1.1t-h2bbff1b_0	None
pip	pkgs/main/win-64::pip-22.3.1-py310haa95532_0	None
python	pkgs/main/win-64::python-3.10.9-h966fe2a_0	None
setuptools	pkgs/main/win-64::setuptools-65.6.3-py310haa95532_0	None

Рисунок 4 – Установка pip

```

(2.14lr) C:\Users\Admin>conda install -n 2.14lr numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 22.9.0
  latest version: 23.1.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\Admin\.conda\envs\2.14lr

added / updated specs:
  - numpy

The following packages will be downloaded:



| package           | build           |        |
|-------------------|-----------------|--------|
| mkl-service-2.4.0 | py310h2bbff1b_0 | 48 KB  |
| mkl_fft-1.3.1     | py310ha0764ea_0 | 136 KB |
| mkl_random-1.2.2  | py310h4ed8f06_0 | 221 KB |
| numpy-1.23.5      | py310h60c9a35_0 | 11 KB  |
| numpy-base-1.23.5 | py310h04254f7_0 | 6.0 MB |
| Total:            |                 | 6.4 MB |



The following NEW packages will be INSTALLED:

blas pkgs/main/win-64::blas-1.0-mkl None
intel-openmp pkgs/main/win-64::intel-openmp-2021.4.0-haa95532_3556 None
mkl pkgs/main/win-64::mkl-2021.4.0-haa95532_640 None
mkl-service pkgs/main/win-64::mkl-service-2.4.0-py310h2bbff1b_0 None
mkl_fft pkgs/main/win-64::mkl_fft-1.3.1-py310ha0764ea_0 None
mkl_random pkgs/main/win-64::mkl_random-1.2.2-py310h4ed8f06_0 None
numpy pkgs/main/win-64::numpy-1.23.5-py310h60c9a35_0 None
numpy-base pkgs/main/win-64::numpy-base-1.23.5-py310h04254f7_0 None
six pkgs/main/noarch::six-1.16.0-pyhd3eb1b0_1 None

Proceed ([y]/n)? y

```

Рисунок 5 – Установка NumPy

```

(2.14lr) C:\Users\Admin>conda install -n 2.14lr pandas
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 22.9.0
  latest version: 23.1.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\Admin\.conda\envs\2.14lr

added / updated specs:
  - pandas

The following packages will be downloaded:



| package          | build           |         |
|------------------|-----------------|---------|
| bottleneck-1.3.5 | py310h9128911_0 | 106 KB  |
| numexpr-2.8.4    | py310hd213c9f_0 | 128 KB  |
| packaging-22.0   | py310haa95532_0 | 68 KB   |
| pandas-1.5.2     | py310h4ed8f06_0 | 10.5 MB |
| pytz-2022.7      | py310haa95532_0 | 210 KB  |
| Total:           |                 | 11.0 MB |



The following NEW packages will be INSTALLED:

bottleneck      pkgs/main/win-64::bottleneck-1.3.5-py310h9128911_0 None
numexpr         pkgs/main/win-64::numexpr-2.8.4-py310hd213c9f_0 None
packaging       pkgs/main/win-64::packaging-22.0-py310haa95532_0 None
pandas          pkgs/main/win-64::pandas-1.5.2-py310h4ed8f06_0 None
python-dateutil pkgs/main/noarch::python-dateutil-2.8.2-pyhd3eb1b0_0 None
pytz           pkgs/main/win-64::pytz-2022.7-py310haa95532_0 None

Proceed ([y]/n)? y

```

Рисунок 6 – Установка Pandas

```
(2.14lr) C:\Users\Admin>conda install -n 2.14lr scipy
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
  current version: 22.9.0
  latest version: 23.1.0
```

Please update conda by running

```
$ conda update -n base -c defaults conda
```

```
## Package Plan ##
```

```
environment location: C:\Users\Admin\.conda\envs\2.14lr
```

```
added / updated specs:
- scipy
```

The following packages will be downloaded:

package	build	
-----	-----	
brotlipy-0.7.0	py310h2bbff1b_1002	335 KB
cffi-1.15.1	py310h2bbff1b_3	239 KB
cryptography-38.0.4	py310h21b164f_0	1.0 MB
idna-3.4	py310haa95532_0	97 KB
pooch-1.4.0	pyhd3eb1b0_0	41 KB
pysocks-1.7.1	py310haa95532_0	28 KB
requests-2.28.1	py310haa95532_0	101 KB
scipy-1.10.0	py310hb9afe5d_0	18.8 MB
urllib3-1.26.14	py310haa95532_0	195 KB
win_inet_pton-1.1.0	py310haa95532_0	9 KB
-----	-----	
Total:		20.8 MB

The following NEW packages will be INSTALLED:

appdirs	pkgs/main/noarch::appdirs-1.4.4-pyhd3eb1b0_0	None
brotlipy	pkgs/main/win-64::brotlipy-0.7.0-py310h2bbff1b_1002	None
cffi	pkgs/main/win-64::cffi-1.15.1-py310h2bbff1b_3	None
charset-normalizer	pkgs/main/noarch::charset-normalizer-2.0.4-pyhd3eb1b0_0	None
cryptography	pkgs/main/win-64::cryptography-38.0.4-py310h21b164f_0	None
fftw	pkgs/main/win-64::fftw-3.3.9-h2bbff1b_1	None
icc_rt	pkgs/main/win-64::icc_rt-2022.1.0-h6049295_2	None
idna	pkgs/main/win-64::idna-3.4-py310haa95532_0	None

Рисунок 7 – Установка Scipy

7. Попробуйте установить менеджером пакетов conda пакет TensorFlow. Возникает ли при этом ошибка? Попробуйте выявить и укажите причину этой ошибки.

```
(2.14lr) C:\Users\Admin>conda install -n 2.14lr tensorflow
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 22.9.0
  latest version: 23.1.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\Admin\.conda\envs\2.14lr

  added / updated specs:
    - tensorflow

The following packages will be downloaded:
```

package	build	size
astunparse-1.6.3	py_0	17 KB
blinker-1.4	py310haa95532_0	22 KB
colorama-0.4.6	py310haa95532_0	32 KB
flatbuffers-2.0.0	h6c2663c_0	1.4 MB
google-pasta-0.2.0	pyhd3eb1b0_0	46 KB
grpcio-1.42.0	py310hc60d5dd_0	1.7 MB
keras-preprocessing-1.1.2	pyhd3eb1b0_0	35 KB
multidict-6.0.2	py310h2bbff1b_0	46 KB
rsa-4.7.2	pyhd3eb1b0_1	28 KB
tensorboard-data-server-0.6.1	py310haa95532_0	17 KB
wrapt-1.14.1	py310h2bbff1b_0	49 KB
yarl-1.8.1	py310h2bbff1b_0	80 KB
Total:		3.4 MB

```

The following NEW packages will be INSTALLED:

 _tfflow_select      pkgs/main/win-64::_tfflow_select-2.3.0-mkl None
 absl-py             pkgs/main/win-64::absl-py-1.3.0-py310haa95532_0 None
 aiohttp             pkgs/main/win-64::aiohttp-3.8.3-py310h2bbff1b_0 None

```

Рисунок 8 – Установка TensorFlow

8. Попробуйте установить пакет TensorFlow с помощью менеджера пакетов pip.

```

(2.141r) C:\Users\Admin>pip install tensorflow
Requirement already satisfied: tensorflow in c:\users\admin\.conda\envs\2.141r\lib\site-packages (2.10.0)
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (1.1.2)
Collecting protobuf<3.20,>=3.9.2
  Downloading protobuf-3.19.6-cp310-cp310-win_amd64.whl (895 kB)
----- 895.7/895.7 kB 4.0 MB/s eta 0:00:00
Requirement already satisfied: setuptools in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (65.6.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (2.0)
Requirement already satisfied: packaging in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (22.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (4.4.0)
Requirement already satisfied: numpy>=1.20 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (1.23.5)
Requirement already satisfied: keras<2.11,>=2.10.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: gast<0.4.0,>=0.2.1 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (0.4.0)
Requirement already satisfied: tensorboard<2.11,>=2.10 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (2.10.0)
Collecting libclang>=13.0.0
  Downloading libclang-15.0.6.1-py2.py3-none-win_amd64.whl (23.2 MB)
----- 23.2/23.2 MB 1.8 MB/s eta 0:00:00
Requirement already satisfied: astunparse>=1.6.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (1.3.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\admin\.conda\envs\2.141r\lib\site-packages (from tensorflow) (3.7.0)

```

Рисунок 9 – Установка TensorFlow

9. Сформируйте файлы requirements.txt и environment.yml .
Проанализируйте содержимое этих файлов.

Файл requirements.txt — это список всех модулей и пакетов Python, которые нужны для полноценной работы программы. Его использование позволяет легко отслеживать весь перечень нужных компонентов, избавляя пользователей от необходимости их ручного поиска и установки. Команда `pip freeze` выводит все установленные в интерпретатор сторонние пакеты.

```

(2.141r) C:\Users\Admin>pip freeze > requirements.txt
(2.141r) C:\Users\Admin>conda env export > environment.yml

```

Рисунок 10 – Создание файлов

```
requirements.txt
1  absl-py @ file:///C:/b/abs_5babsu7y5x/croot/absl-py_1666362945682/work
2  aiohttp @ file:///C:/b/abs_c4zmy2l696/croot/aiohttp_1670009573673/work
3  aiosignal @ file:///tmp/build/80754af9/aiosignal_1637843061372/work
4  appdirs==1.4.4
5  astunparse==1.6.3
6  async-timeout @ file:///C:/b/abs_43ozhz2a8q/croots/recipe/async-timeout_1664876362767/work
7  attrs @ file:///C:/b/abs_09s3v775ra/croot/attrs_1668696195628/work
8  blinker==1.4
9  Bottleneck @ file:///C:/Windows/Temp/abs_3198ca53-903d-42fd-87b4-03e6d03a8381yfwsuve8/croots/recipe/bottleneck_1657175565403/work
10 brotli==0.7.0
11 cachetools @ file:///tmp/build/80754af9/cachetools_1619597386817/work
12 certifi @ file:///C:/b/abs_85o_6fm0se/croot/certifi_1671487778835/work/certifi
13 cffi @ file:///C:/b/abs_49n3v2hyhr/croot/cffi_1670423218144/work
14 charset-normalizer @ file:///tmp/build/80754af9/charset-normalizer_1630003229654/work
15 click @ file:///C:/ci/click_1646056762388/work
16 colorama @ file:///C:/b/abs_a9ozg0l032/croot/colorama_1672387194846/work
17 cryptography @ file:///C:/b/abs_b7d7drzbky/croot/cryptography_1673298763653/work
18 flatbuffers @ file:///home/ktietz/cip/python-flatbuffers_1634039120618/work
19 flit core @ file:///opt/conda/conda-bld/flit-core_1644941570762/work/source/flit_core
20 frozenlist @ file:///C:/b/abs_2bb5uzqhsi/croot/frozenlist_1670004511812/work
21 gast @ file:///Users/ktietz/demo/mc3/conda-bld/gast_1628588903283/work
22 google-auth @ file:///opt/conda/conda-bld/google-auth_1646735974934/work
23 google-auth-oauthlib @ file:///tmp/build/80754af9/google-auth-oauthlib_1617120569401/work
24 google-pasta @ file:///Users/ktietz/demo/mc3/conda-bld/google-pasta_1630577991354/work
25 grpcio @ file:///C:/ci/grpcio_1655475084232/work
26 h5py @ file:///C:/ci/h5py_1659089830381/work
27 idna @ file:///C:/b/abs_bdhbebrloa/croot/idna_1666125572046/work
28 keras @ file:///C:/Users/builder/adi Pietro/mc3/tf210/conda-bld/keras_1669760570649/work/keras-2.10.0-py2.py3-none-any.whl
29 Keras-Preprocessing @ file:///tmp/build/80754af9/keras-preprocessing_1612283640596/work
30 libclang==15.0.6.1
31 Markdown @ file:///C:/b/abs_98lv_ucina/croot/markdown_1671541919225/work
32 MarkupSafe @ file:///C:/ci/markupsafe_1654508036328/work
33 mkl-fft==1.3.1
34 mkl-random @ file:///C:/ci_310/mkl_random_1643050563308/work
35 mkl-service==2.4.0
36 multidict @ file:///C:/b/abs_6cx_8w3cv2/croot/multidict_1665674238352/work
37 numexpr @ file:///C:/b/abs_a7kbak88hk/croot/numexpr_1668713882979/work
38 numpy @ file:///C:/b/abs_datssh7cer/croot/numpy_and_numpy_base_1672336199388/work
```

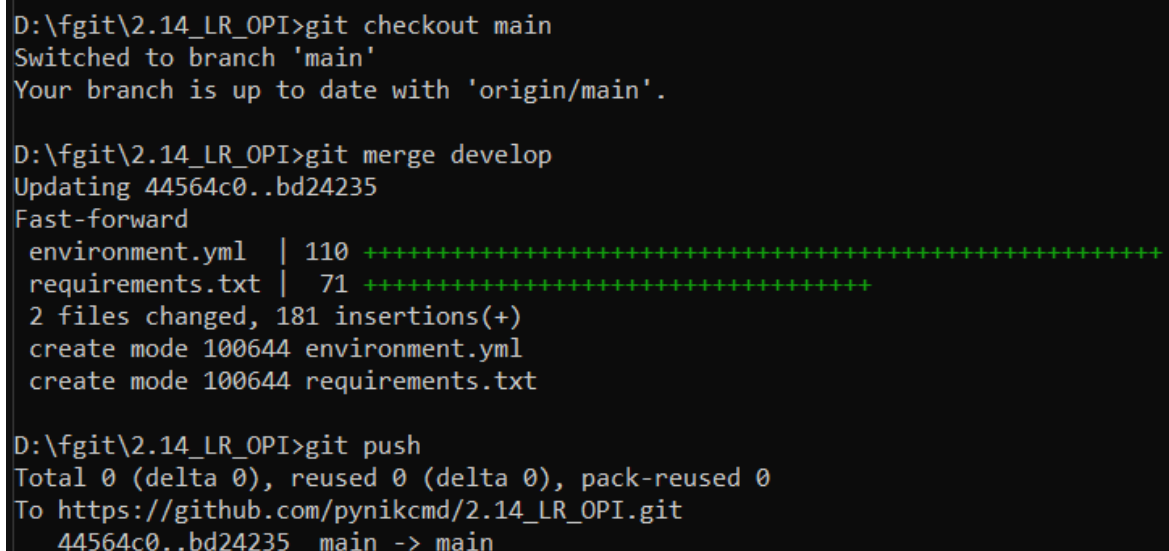
Рисунок 11 – Содержимое файла requirements.txt



```
environment.yml
1  name: 2.14lr
2  channels:
3    - defaults
4  dependencies:
5    - _tflow_select=2.3.0=mkl
6    - absl-py=1.3.0=py310haa95532_0
7    - aiohttp=3.8.3=py310h2bbff1b_0
8    - aiosignal=1.2.0=pyhd3eb1b0_0
9    - appdirs=1.4.4=pyhd3eb1b0_0
10   - astunparse=1.6.3=py_0
11   - async-timeout=4.0.2=py310haa95532_0
12   - attrs=22.1.0=py310haa95532_0
13   - blas=1.0=mkl
14   - blinker=1.4=py310haa95532_0
15   - bottleneck=1.3.5=py310h9128911_0
16   - brotliipy=0.7.0=py310h2bbff1b_1002
17   - bzip2=1.0.8=he774522_0
18   - ca-certificates=2023.01.10=haa95532_0
19   - cachetools=4.2.2=pyhd3eb1b0_0
20   - certifi=2022.12.7=py310haa95532_0
21   - cffi=1.15.1=py310h2bbff1b_3
22   - charset-normalizer=2.0.4=pyhd3eb1b0_0
23   - click=8.0.4=py310haa95532_0
24   - colorama=0.4.6=py310haa95532_0
25   - cryptography=38.0.4=py310h21b164f_0
26   - fftw=3.3.9=h2bbff1b_1
27   - flatbuffers=2.0.0=h6c2663c_0
28   - flit-core=3.6.0=pyhd3eb1b0_0
29   - frozenlist=1.3.3=py310h2bbff1b_0
30   - gast=0.4.0=pyhd3eb1b0_0
```

Рисунок 12 – Содержимое файла environment.yml

10. Зафиксируйте сделанные изменения в репозитории.



```
D:\fgit\2.14_LR_OPI>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

D:\fgit\2.14_LR_OPI>git merge develop
Updating 44564c0..bd24235
Fast-forward
 environment.yml | 110 +++++
 requirements.txt | 71 +++++
 2 files changed, 181 insertions(+)
 create mode 100644 environment.yml
 create mode 100644 requirements.txt

D:\fgit\2.14_LR_OPI>git push
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/pynikcmd/2.14_LR_OPI.git
 44564c0..bd24235  main -> main
```

Рисунок 13 – Сохранение выполненной работы

Вопросы для защиты работы

1. Каким способом можно установить пакет Python, не входящий в стандартную библиотеку?

Существует Python Package Index (PyPI) – это репозиторий, открытый для всех разработчиков, в нём можно найти пакеты для решения практических задач.

2. Как осуществить установку менеджера пакетов pip?

Pip – это консольная утилита (без графического интерфейса). После того, как вы её скачаете и установите, она пропишется в PATH и будет доступна для использования. Чтобы установить утилиту pip, нужно скачать скрипт get-pip.py.

3. Откуда менеджер пакетов pip по умолчанию устанавливает пакеты?

По умолчанию в Linux Pip устанавливает пакеты в /usr/local/lib/python2.7/dist-packages. Использование virtualenv или --user во время установки изменит это местоположение по умолчанию. Важный момент: по умолчанию pip устанавливает пакеты глобально. Это может привести к конфликтам между версиями пакетов.

4. Как установить последнюю версию пакета с помощью pip?

```
pip install projectname
```

5. Как установить заданную версию пакета с помощью pip?

```
pip install projectname==3.2
```

6. Как установить пакет из git репозитория (в том числе GitHub) с помощью pip?

```
pip install -e git+https://gitrepo.com/...
```

7. Как установить пакет из локальной директории с помощью pip?

```
pip install путь
```

8. Как удалить установленный пакет с помощью pip?

```
pip uninstall projectname
```

9. Как обновить установленный пакет с помощью pip?

```
pip install --upgrade projectname
```

10. Как отобразить список установленных пакетов с помощью pip?

```
pip list
```

11. Каковы причины появления виртуальных окружений в языке Python?

Если разработчик работает над проектом не один, а с командой, ему нужно передавать и получать список зависимостей, а также обновлять их на своем компьютере таким образом, чтобы не нарушалась работа других его проектов. Значит нам нужен механизм, который вместе с обменом проектами быстро устанавливал бы локально и все необходимые для них пакеты, при этом не мешая работе других проектов. Идея виртуального окружения родилась раньше, чем была реализована стандартными средствами Python. Попыток было несколько, но в основу PEP 405 легла утилита `virtualenv` Яна Бикинга. Были проанализированы возникающие при работе с ней проблемы. После этого в работу интерпретатора Python версии 3.3 добавили их решения. Так был создан встроенный в Python

модуль `venv`, а утилита `virtualenv` теперь дополнительно использует в своей работе и его.

Как работает виртуальное окружение: в отдельной папке создаётся неполная копия выбранной установки Python. Это копия является просто набором файлов (например, интерпретатора или ссылки на него), утилит для работы с собой и нескольких пакетов (в том числе `pip`). Стандартные пакеты при этом не копируются.

12. Каковы основные этапы работы с виртуальными окружениями?

1) Создаём через утилиту новое виртуальное окружение в отдельной папке для выбранной версии интерпретатора Python.

2) Активируем ранее созданное виртуального окружения для работы.

3) Работаем в виртуальном окружении, а именно управляем пакетами используя `pip` и запускаем выполнение кода.

4) Деактивируем после окончания работы виртуальное окружение.

5) Удаляем папку с виртуальным окружением, если оно нам больше не нужно.

13. Как осуществляется работа с виртуальными окружениями с помощью `venv`?

```
python3 -m venv <путь к папке виртуального окружения>
```

Создадим виртуальное окружение в папке проекта. Для этого перейдём в корень любого проекта на Python ≥ 3.3 и дадим команду:

```
$ python3 -m venv env
```

После её выполнения создастся папка `env` с виртуальным окружением. Чтобы активировать виртуальное окружение под Windows нужно дать команду:

```
> env\\Scripts\\activate  
просто под windows мы вызываем скрипт активации напрямую.
```

После активации приглашение консоли изменится. В его начале в круглых скобках будет отображаться имя папки с виртуальным окружением.

При размещении виртуального окружения в папке проекта стоит позаботиться об его исключении из репозитория системы управления версиями. Для этого, например, при использовании Git нужно добавить папку в файл .gitignore. Это делается для того, чтобы не засорять проект разными вариантами виртуального окружения.

```
$ python3 -m venv /home/user/envs/project1_env
```

Чтобы переключиться с одного окружения на другое нам нужно выполнить команду деактивации и команду активации другого виртуального окружения.

```
$ deactivate  
$ source /home/user/envs/project1_env2/bin/activate
```

14. Как осуществляется работа с виртуальными окружениями с помощью virtualenv?

Для начала пакет нужно установить. Установку можно выполнить командой:

```
# для python 3  
python3 -m pip install virtualenv  
  
# для единственного python  
python -m pip install virtualenv
```

Создание виртуального окружения с утилитой virtualenv отличается от стандартного. Например, создание в текущей папке виртуального

окружения для интерпретатора доступного через команду `python3` с названием папки окружения `env`:

```
virtualenv -p python3 env
```

```
> env\\Scripts\\activate
```

```
(env) > deactivate
```

Активация и деактивация такая же, как у стандартной утилиты Python.

15. Изучите работу с виртуальными окружениями `pipenv`. Как осуществляется работа с виртуальными окружениями `pipenv`?

Грубо говоря, `pipenv` можно рассматривать как симбиоз утилит `pip` и `venv` (или `virtualenv`), которые работают вместе, пряча многие неудобные детали от конечного пользователя. Помимо этого `pipenv` ещё умеет вот такое:

- автоматически находить интерпретатор Python нужной версии (находит даже интерпретаторы, установленные через `pyenv` и `asdf`!);
- запускать вспомогательные скрипты для разработки;
- загружать переменные окружения из файла `.env`;
- проверять зависимости на наличие известных уязвимостей.

Стоит сразу оговориться, что если вы разрабатываете библиотеку (или что-то, что устанавливается через `pip`, и должно работать на нескольких версиях интерпретатора), то `pipenv` — не ваш путь. Этот инструмент создан в первую очередь для разработчиков конечных приложений (консольных утилит, микросервисов, веб-сервисов). Формат хранения зависимостей подразумевает работу только на одной конкретной версии интерпретатора (это имеет смысл для конечных приложений, но для библиотек это, как правило, не приемлемо).

Для разработчиков библиотек существует другой прекрасный инструмент — poetry.

Установка на Windows, самый простой способ — это установка в домашнюю директорию пользователя:

```
$ pip install --user pipenv
```

Теперь проверим установку:

```
$ pipenv --version
```

```
pipenv, version 2018.11.26
```

Если вы получили похожий вывод, значит, всё в порядке.

Инициализация проекта

Давайте создадим простой проект под управлением pipenv.

Подготовка:

```
$ mkdir pipenv_demo
```

```
$ cd pipenv_demo
```

Создать новый проект, использующий конкретную версию Python можно вот такой командой:

```
$ pipenv --python 3.8
```

Если же вам не нужно указывать версию так конкретно, то есть

шорткаты: # Создает проект с Python 3, версию выберет автоматически.

```
$ pipenv --three
```

```
# Аналогично с Python 2.
```

```
# В 2020 году эта опция противопоказана.
```

```
$ pipenv --two
```

После выполнения одной из этих команд, pipenv создал файл Pipfile и виртуальное окружение где-то в заранее определенной директории (по умолчанию вне директории проекта).

```
$ cat Pipfile [[source]] name = "pypi"
```

```
url = "https://pypi.org/simple" verify_ssl = true[dev-packages]
```

```
[packages] [requires] python_version = "3.8"
```

Это минимальный образец Pipfile. В секции `[[source]]` перечисляются индексы пакетов — сейчас тут только PyPI, но может быть и ваш собственный индекс пакетов. В секциях `[packages]` и `[dev-packages]` перечисляются зависимости приложения — те, которые нужны для непосредственной работы приложения (минимум), и те, которые нужны для разработки (запуск тестов, линтеры и прочее). В секции `[requires]` указана версия интерпретатора, на которой данное приложение может работать.

Если вам нужно узнать, где именно `pipenv` создал виртуальное окружение (например, для настройки IDE), то сделать это можно вот так:

```
$ pipenv --py
```

```
/Users/and-semakin/.local/share/virtualenvs/pipenv_demo 1dgGUSFy/bin/python
```

Управление зависимостями через `pipenv` Теперь давайте установим в проект первую зависимость. Делается это при помощи команды `pipenv install`:

```
$ pipenv install requests
```

Давайте посмотрим, что поменялось в Pipfile (здесь и дальше я буду сокращать вывод команд или содержимое файлов при помощи ...):

```
$ cat Pipfile
```

```
...
```

```
[packages] requests = "*"

...
```

В секцию `[packages]` добавилась зависимость `requests` с версией `*` (версия нефиксирована).

А теперь давайте установим зависимость, которая нужна для разработки, например, восхитительный линтер `flake8`, передав флаг `--dev` в ту

же команду `install`:

```
$ pipenv install --dev flake8
```

```
$ cat Pipfile
```

```
...
```

```
[dev-packages] flake8 = "*"
```

```
...
```

Теперь можно увидеть всё дерево зависимостей проекта при помощи команды `pipenv graph`:

```
$ pipenv graph flake8==3.7.9
```

```
- entrypoints [required: >=0.3.0,<0.4.0, installed: 0.3]
- mccabe [required: >=0.6.0,<0.7.0, installed: 0.6.1]
- pycodestyle [required: >=2.5.0,<2.6.0, installed: 2.5.0]
- pyflakes [required: >=2.1.0,<2.2.0, installed: 2.1.1] requests==2.23.0
- certifi [required: >=2017.4.17, installed: 2020.4.5.1]
- chardet [required: >=3.0.2,<4, installed: 3.0.4]
- idna [required: >=2.5,<3, installed: 2.9]
- urllib3 [required: >=1.21.1,<1.26,!1.25.1,!1.25.0, installed: 1.25.9]
```

Это бывает полезно, чтобы узнать, что от чего зависит, или почему в вашем виртуальном окружении есть определённый пакет. Также, пока мы устанавливали пакеты, `pipenv` создал `Pipfile.lock`, но тот файл длинный и не интересный, поэтому показывать содержимое я не буду.

Удаление и обновление зависимостей происходит при помощи команд `pipenv uninstall` и `pipenv update` соответственно. Работают они довольно интуитивно, но если возникают вопросы, то вы всегда можете получить справку при помощи флага `--help`:

```
$ pipenv uninstall --help
```

```
$ pipenv update --help
```

Управление виртуальными окружениями

Давайте удалим созданное виртуальное окружение:

```
$ pipenv --rm
```

И представим себя в роли другого разработчика, который только присоединился к вашему проекту. Чтобы создать виртуальное окружение и установить в него зависимости нужно выполнить следующую команду:

```
$ pipenv sync --dev
```

Эта команда на основе Pipfile.lock воссоздаст точно то же самое виртуальное окружение, что и у других разработчиков проекта.

Если же вам не нужны dev-зависимости (например, вы разворачиваете ваш проект на продакшн), то можно не передавать флаг --dev:

```
$ pipenv sync
```

Чтобы "войти" внутрь виртуального окружения, нужно выполнить:

```
$ pipenv shell (pipenv_demo) $
```

В этом режиме будут доступны все установленные пакеты, а имена python и pip будут указывать на соответствующие программы внутри виртуального окружения.

Есть и другой способ запускать что-то внутри виртуального окружения

безсоздания нового шелла:

```
# это запустит REPL внутри виртуального окружения
```

```
$ pipenv run python
```

```
# а вот так можно запустить какой-нибудь файл
```

```
$ pipenv run python script.py
```

```
# а так можно получить список пакетов внутри виртуального окружения
```

```
$ pipenv run pip freeze
```

16. Каково назначение файла requirements.txt ? Как создать этот файл? Какой он имеет формат?

Просмотреть список зависимостей мы можем командой: `pip freeze > requirements.txt`

Имя файла хранения зависимостей `requirements.txt` выбрано не зря. Оно является стандартной договоренностью и используется некоторыми утилитами автоматически.

Установка пакетов из файла зависимостей в новом виртуальном окружении также выполняется одной командой:

```
pip install -r requirements.txt
```

Все пакеты, которые вы установили перед выполнением команды и предположительно использовали в каком-либо проекте, будут перечислены в файле с именем «`requirements.txt`». Кроме того, будут указаны их точные версии.

17. В чем преимущества пакетного менеджера `conda` по сравнению с пакетным менеджером `pip`?

Основная проблема заключается в том, что `pip`, `easy_install` и `virtualenv` ориентированы на Python. Эти инструменты игнорируют библиотеки зависимостей, реализованные с использованием других языков. Например, XSLT, HDF5, MKL и другие, которые не имеют `setup.py` в исходном коде и не устанавливают файлы в директорию `site-packages`. Conda же способна управлять пакетами как для Python, так и для C/ C++, R, Ruby, Lua, Scala и других. Conda устанавливает двоичные файлы, поэтому работу по компиляции пакета самостоятельно выполнять не требуется (по сравнению с `pip`).

Существуют также некоторые различия, если вы заинтересованы в создании собственных пакетов. Например, `pip` создан на основе `setuptools`, тогда как `conda` использует свой собственный формат, который имеет некоторые преимущества (например, статическая компиляция пакета).

18. В какие дистрибутивы Python входит пакетный менеджер conda?
Anaconda, miniconda и PyCharm.

19. Как создать виртуальное окружение conda?

1. Начиная проект, создайте чистую директорию и дайте ей понятное короткое имя.

Для Windows, если используется дистрибутив Anaconda, то необходимо вначале запустить консоль Anaconda Powershell Prompt. Делается это из системного меню, посредством выбора следующих пунктов: Пуск Anaconda3 (64-bit) Anaconda Powershell Prompt (Anaconda3). В результате будет отображено окно консоли, показанное на рисунке. Обратите на имя виртуального окружения по умолчанию, которым в данном случае является base. В этом окне необходимо ввести следующую последовательность команд:

```
mkdir %PROJ_NAME% cd %PROJ_NAME% copy NUL > main.py
```

Здесь PROJ_NAME - это переменная окружения, в которую записано имя проекта. Допускается не использовать переменные окружения, а использовать имя проекта вместо \$PROJ_NAME или %PROJ_NAME% .

20. Как активировать и установить пакеты в виртуальное окружение conda?

```
conda create -n %PROJ_NAME% python=3.7 conda activate  
%PROJ_NAME%
```

Установите пакеты, необходимые для реализации проекта. conda install django,pandas

21. Как деактивировать и удалить виртуальное окружение conda?

Для Windows необходимо использовать следующую команду:

```
conda deactivate
```

Если вы хотите удалить только что созданное окружение, выполните:

```
conda remove -n $PROJ_NAME
```

22. Каково назначение файла `environment.yml` ? Как создать этот файл?

Файл `environment.yml` позволит воссоздать окружение в любой нужный момент. Достаточно набрать:

```
conda env create -f environment.yml
```

23. Как создать виртуальное окружение conda с помощью файла `environment.yml`?

```
conda env export > environment.yml
```

24. Самостоятельно изучите средства IDE PyCharm для работы с виртуальными окружениями conda. Опишите порядок работы с виртуальными окружениями conda в IDE PyCharm.

Создавайте отдельное окружение Conda и устанавливайте только нужные библиотеки для каждого проекта. PyCharm позволяет легко создавать и выбирать правильное окружение.

25. Почему файлы `requirements.txt` и `environment.yml` должны храниться в репозитории git?

Предоставляет доступ другим пользователям к файлам