

Заметки по машинному обучению и анализу данных

Подвойский Александр alexander.podvoyskiy@ipd.zyfra.com*

Здесь будут собираться заметки по вопросам, касающимся машинного обучения, интеллектуального анализа данных, программирования (Python, Scala etc.), прикладной математики, а также по прочим сопряженным вопросам так или иначе, затрагивающим работу с данными

Содержание

1	Инструмент управления git-публикациями pre-commit	1
1.1	Краткое описание	1
1.2	Порядок работы	4
1.3	Полезные ресурсы	4
2	Библиотека csvkit для работы с большими csv-файлами в командной оболочке	5
2.1	Краткое описание	5
2.2	Примеры использования	5
2.3	Полезные ресурсы	6
3	Сервис статического анализа кодовой базы deepsource	6
3.1	Краткое описание	6
3.2	Порядок работы	6
3.3	Полезные ресурсы	9

1. Инструмент управления git-публикациями pre-commit

1.1. Краткое описание

`pre-commit` – это простой удобный инструмент управления git-хуками. Хук представляет собой пакет¹, реализующий некоторую логику работы с зафиксированными изменениями кодовой базы до публикации этих изменений на удаленном сервере.

`pre-commit` поддерживает возможность создавать пользовательские хуки <https://pre-commit.com/#new-hooks>.

Установить инструмент можно, как обычно, с помощью менеджера пакетов `pip`

```
pip install pre-commit
pre-commit install # установить pre-commit в git-хуки
```

После установки пакета `pre-commit` в командной оболочке будет доступна утилита командной строки с тем же именем.

Для того чтобы при фиксации изменений кодовой базы (`git commit`) запускалась цепочка проверок, следует в корне проекта разместить конфигурационный файл `.pre-commit-config.yaml`.

*Комментарии и предложения приветствуются. Поругать автора можно по указанному адресу

¹Поддерживаются различные технологии: `bash`, `Python`, `dotenv`, `docker`, `ruby` etc.

Типичный конфигурационный файл `pre-commit` управления `git`-хуками выглядит следующим образом

`.pre-commit-config.yaml`

```
repos:
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: v4.0.1
  hooks:
    # проверяет наличие переноса строки в конце всех текстовых файлов
    - id: end-of-file-fixer
    # предупреждает о добавлении больших файлов в Git
    - id: check-added-large-files
    # предупреждает о сохранении файлов с UTF-8 BOM
    - id: fix-byte-order-marker
    # предотвращает сохранение приватных ключей
    - id: detect-private-key
    # убивает пробелы в конце строки
    - id: trailing-whitespace
    # проверяет на предмет расположения docstring после кода
    - id: check-docstring-first
    # проверяет файлы на предмет конфликтующих строк при операции слияния
    - id: check-merge-conflict
    # проводит синтаксический анализ yaml-файлов
    - id: check-yaml
    # проводит синтаксический анализ toml-файлов
    - id: check-toml
    # проводит синтаксический анализ json-файлов
    - id: check-json
- repo: https://github.com/pre-commit/mirrors-isort
  rev: f0001b2 # Use the revision sha / tag you want to point at
  hooks:
    - id: isort
      args: ["--profile", "black"]
- repo: https://github.com/psf/black
  rev: 21.7b0
  hooks:
    - id: black
      language_version: python3
- repo: https://github.com/asottile/yesqa
  rev: v1.1.0
  hooks:
    - id: yesqa
      additional_dependencies:
        - flake8-bugbear==20.1.4
        - flake8-builtins==1.5.2
        - flake8-comprehensions==3.2.2
        - flake8-tidy-imports==4.1.0
        - flake8==3.7.9
- repo: https://github.com/asottile/pyupgrade
  rev: v2.7.3
  hooks:
    - id: pyupgrade
      args: ['--py37-plus']
- repo: https://github.com/pre-commit/pygrep-hooks
  rev: v1.5.1
  hooks:
    - id: python-check-mock-methods
    - id: python-use-type-annotations
```

```
ci:
  autoupdate_commit_msg: 'chore: pre-commit autoupdate'
```

Теперь при каждой операции `git commit` будет запускаться цепочка проверок. Однако при желании эту цепочку можно запустить и без фиксации изменений, просто набрав в командной оболочке `pre-commit run --all-files` (см. рис. 1).

```
$ pre-commit run --all-files
[INFO] Initializing environment for https://github.com/pre-commit/pre-commit-hooks.
[INFO] Initializing environment for https://github.com/psf/black.
[INFO] Installing environment for https://github.com/pre-commit/pre-commit-hooks.
[INFO] Once installed this environment will be reused.
[INFO] This may take a few minutes...
[INFO] Installing environment for https://github.com/psf/black.
[INFO] Once installed this environment will be reused.
[INFO] This may take a few minutes...
Check Yaml.....Passed
Fix End of Files.....Passed
Trim Trailing Whitespace.....Failed
- hook id: trailing-whitespace
- exit code: 1

Files were modified by this hook. Additional output:

Fixing sample.py

black.....Passed
```

Рис. 1. Сеанс `pre-commit run`

Для управления настройками отдельных хуков (`flake8`, `black` и т.д.) в корне проекта можно разместить соответствующие конфигурационные файлы.

Например, для `flake8`

`.flake8`

```
[flake8]
ignore =
    E402, W503 # для того, чтобы flake8 пропускал сообщения с этими метками
```

Для хука `isort`

`.isort.cfg`

```
[tool.isort]
profile = "black"
multi_line_output = 3
include_trailing_comma = True
force_grid_wrap = 0
use_parentheses = True
ensure_newline_before_comments = True
line_length = 119
```

Для хука `black`

`pyproject.toml`

```
# Example configuration for Black.

# NOTE: you have to use single-quoted strings in TOML for regular expressions.
# It's the equivalent of r-strings in Python. Multiline strings are treated as
# verbose regular expressions by Black. Use [ ] to denote a significant space
# character.

[tool.black]
line-length = 79
target-version = ['py36', 'py37', 'py38']
include = '\.pyi?$'
exclude = '''
/(
    \.eggs
  | \.git
  | \.hg
  | \.mypy_cache
  | \.tox
  | \.venv
  | _build
  | buck-out
  | build
  | dist
)/
'''
```

1.2. Порядок работы

В сухом остатке простейший шаблон работы с `pre-commit` выглядит так

- либо цепочка проверок запускается через командную оболочку

```
pre-commit --version # pre-commit 2.14.0
pre-commit run --color always --all-files # чтобы запустить все хуки
pre-commit run <hook_id> # если нужно запустить какой-то конкретный хук
pre-commit clean # очищает кэш
pre-commit gc # удаляет неиспользуемые репозитории кэш-каталога; рекомендуется выполнять эт
у команду время от времени
```

- либо автоматически «за кадром» при попытке фиксации изменений с помощью `git commit`

Убедиться в том, что `Git` использует не сценарий `pre-commit` по умолчанию, а тот сценарий, который создан пакетом `pre-commit` можно так

```
cat .git/hooks/pre-commit | sed -n "/gener.*p/" # File generated by pre-commit: https://pre-
commit.com
```

1.3. Полезные ресурсы

Сайт проекта `pre-commit`: <https://pre-commit.com/#plugins>.

Каталог хуков: <https://pre-commit.com/hooks.html>.

2. Библиотека csvkit для работы с большими csv-файлами в командной оболочке

2.1. Краткое описание

Иногда возникает необходимость *быстро* провести разведочный анализ данных, представленных в виде «больших» (несколько сотен мегабайт) csv-файлов без необходимости привлекать специализированные библиотеки типа `pandas`, `dask`, `polars` и пр.

Утилита `csvkit` как раз представляет собой такой инструмент командной строки.

Установить можно, как обычно с помощью, `pip`

```
pip install csvkit
```

После установки пакета в командной оболочке будут доступны следующие утилиты

- `csvlook`: отвечает за «human-readable»-представление csv-файлов,
- `csvcut`: фильтрует и усекает csv-файлы (работает по аналогии с утилитой Linux `cut`),
- `in2csv`: преобразует различные табличные форматы, включая `*.xls(x)`, `*.geojson`, `*.dbf` и пр., в `*.csv`,
- `csvstat`: возвращает описательные статистики для выбранных столбцов,
- `csvgrep`: отбирает строки, которые отвечают заданному условию или регулярному выражению,
- `csvsort`: сортирует csv-файлы (работает также как и Linux-аналог `sort`),
- `csvjoint`: соединяет csv-файлы «горизонтально»,
- `csvstack`: соединяет csv-файлы «вертикально»,
- `csvsql`: выполняет SQL-запрос на csv-файле.

2.2. Примеры использования

Преобразование табличных форматов с заданной схемой в csv-файл

```
# преобразовать json-файл в csv-файл в потоке
curl https://api.github.com/repos/.../issues?state=open | in2csv --format json -v
# простое преобразование базы данных *.dbf в csv-файл
in2csv examples/testdbf.dbf
```

Рендеринг csv-файлов

```
# рендеринг 3-его и 1-ого столбцов набора данных
csvcut -c 3,1 filename.csv | head -n 5 | csvlook
```

Работа с подвыборками

```
# извлечь 3-ий и 5-ый столбец
csvcut -c 3,5 filename.csv
# извлечь столбцы с заданными именами
csvcut -c TOTAL,"State Name" filename.csv
```

Описательные статистики

```
# вернуть уникальные значения для 2-ого и 6-ого столбцов
csvcut -c 2,6 | csvstat --freq titanic.csv
# вернуть число уникальных значений в 6 столбце
csvstat -c 6 --unique titanic.csv
```

Фильтрация по строкам

```
# выбрать из 5-ого столбца строки, в которых встречаются имена, содержащие подстроку "Will"
csvgrep -c 5 -r ".*Will.*" titanic.csv
```

Выполнение SQL-запросов над csv-файлами

```
# сложить в стек два csv-файла и выбрать все столбцы
csvstack csv_file_part1.csv csv_file_part2.csv | csvsql --query "select * from stdin"
# выполнить внутреннее объединение двух csv-файлов по столбцу species, затем сгруппировать по нем
у и подсчитать агрегат
csvsql --query "select m.usda_id, avg(i.sepal_length) as mean_sepal_length from iris as i join
    irismeta as m on (i.species = m.species) group by m.species" examples/iris.csv examples/
    irismeta.csv
```

2.3. Полезные ресурсы

Документация проекта csvkit: <https://csvkit.readthedocs.io/en/latest/index.html>.

3. Сервис статического анализа кодовой базы deepsource

3.1. Краткое описание

deepsource – это сервис автоматизации статического анализа кода. Для открытых исследовательских проектов сервис не требует никакой платы, но для коммерческих проектов придется купить подписку.

Получить доступ к сервису можно через GitHub, GitLab или Bitbucket. Создав учетную запись <https://deepsource.io/docs/setup-analysis> DeepSource.io останется только развернуть приложение DeepSource на сервисе управления репозиториями. Например, в случае GitHub приглашение будет выглядеть как показано на рис. 2. Здесь нужно указать для каких репозиториях будет проводиться анализ, а затем нажать кнопку «Install».

По завершении DeepSource можно будет использовать как дашборд (рис. 3). Управлять процедурой анализа можно как показано в источнике <https://deepsource.io/docs/setup-analysis#activate-analysis>.

3.2. Порядок работы

DeepSource проведет по всем шагам – от создания учетной записи и до настройки анализаторов кода – и в итоге в корне репозитория будет создан конфигурационный файл `.deepsource.toml` (содержание может быть другим в зависимости от пользовательских настроек)

`.deepsource.toml`

```
version = 1

test_patterns = [
    'tests/**'
]

[[analyzers]]
name = "python" # анализаторы для Python
enabled = true
runtime_version = "3.x.x"

[analyzers.meta]
```

```
max_line_length = 79

[[analyzers]] # анализаторы на покрытие
name = "test-coverage"
enabled = true
```

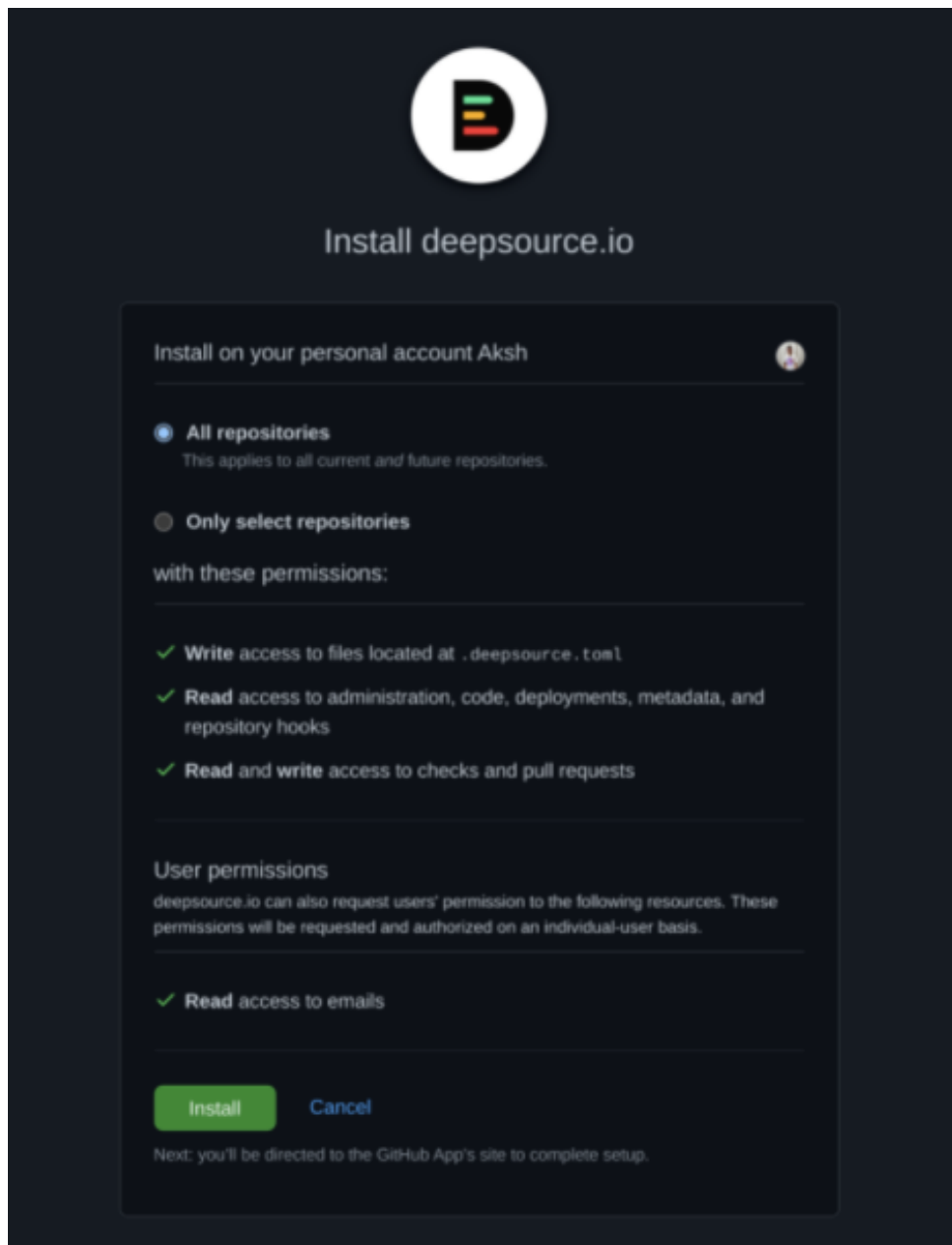


Рис. 2. Установка приложения DeepSource на GitHub

Аналогичным образом можно включать блоки для других поддерживаемых технологий и языков программирования. Например, для Docker

.deepsource (для Docker)

```
version = 1

[[analyzers]]
name = "docker"
enabled = true

[analyzers.meta]
```

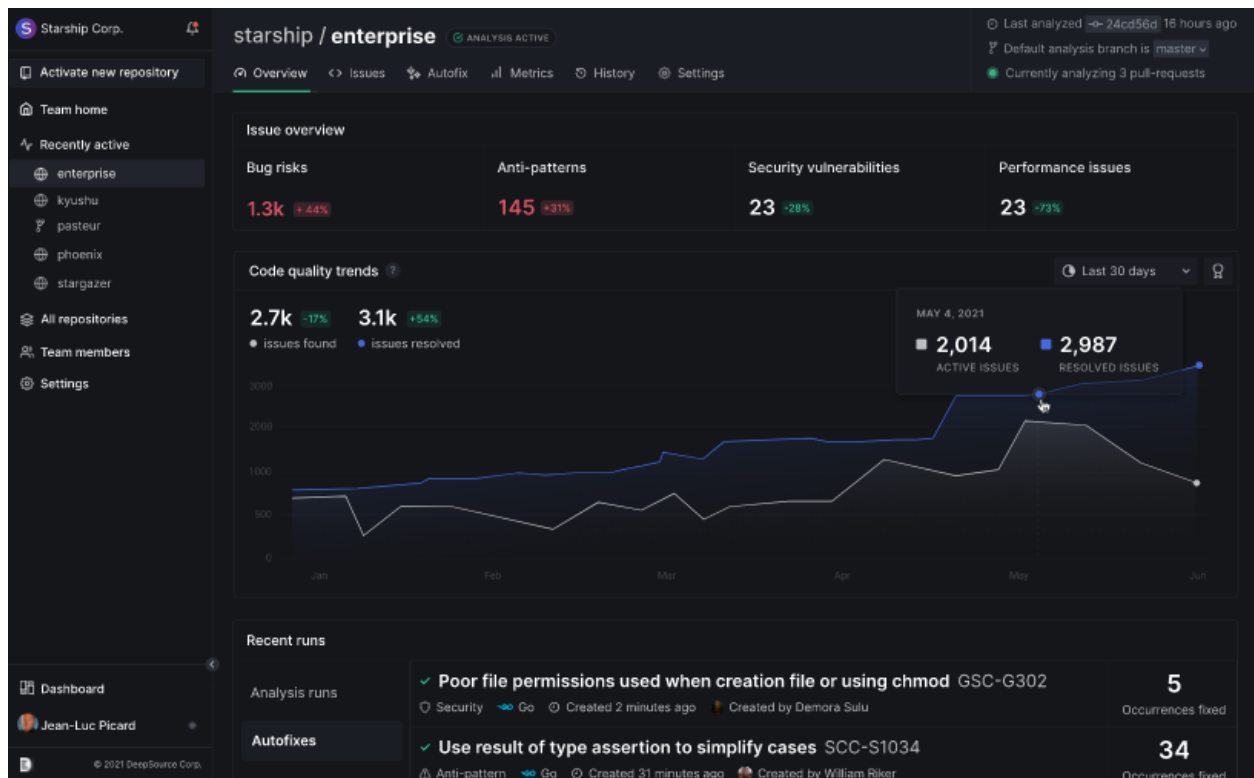


Рис. 3. Дешборд DeepSource

```
dockerfile_paths = [
    "dockerfile_dev",
    "dockerfile_prod"
]

trusted_registries = [
    "my-registry.com",
    "docker.io"
]
...
```

Для SQL

.deepsources (для SQL)

```
version = 1

[[analyzers]]
name = "sql"
enabled = true

[analyzers.meta]
max_line_length = 100
tab_space_size = 4
indent_unit = "tab"
comma_style = "trailing"
capitalisation_policy = "consistent"
allow_scalar = true
single_table_references = "consistent"
```

Для Scala

.deepsources (для Scala)


```
version = 1

test_patterns = [
    "test/**",
    "*_test.scala"
]

exclude_patterns = [
    "vendor/**",
    "**/examples/**"
]

[[analyzers]]
name = "scala"
enabled = true
```

3.3. Полезные ресурсы

Документация проекта `deepsource.io`: <https://deepsource.io/>