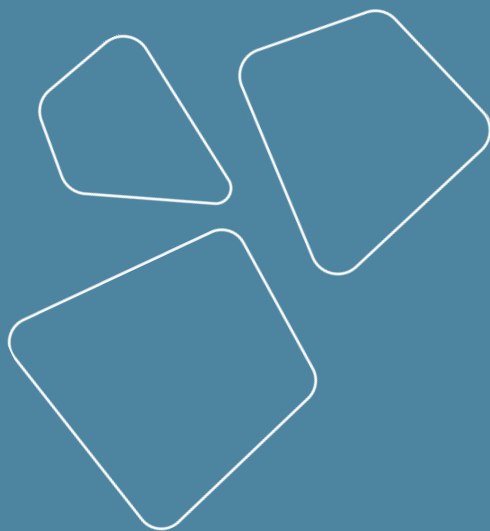# Управление кодом и git servers

Vladislav Goncharenko

MIPT, MSU, fall 2023
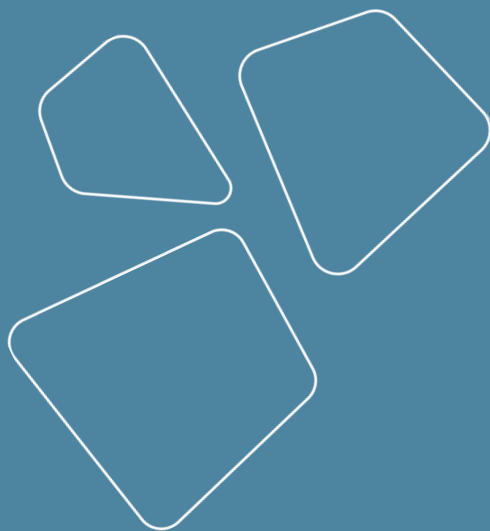
girafe
ai

Recap

- Зачем нужен MLOps
- Стандарты разработки Data Science проектов
- Что такое MLOps
- Подзадачи MLOps
- Обзор решений от крупных компаний
- Проблемы open source стека

- Пакетирование в Питоне
- Управление зависимостями
  - poetry
- Jupyter server с паролем

Outline

- Хранение кода
- Git servers
- Работа в команде
  - Дистрибуция кода
- CI/CD
- Структура кодовой базы
  - Форматирование кода
  - Inline документация

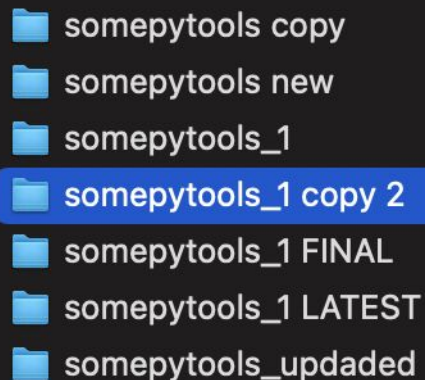- pre-commit
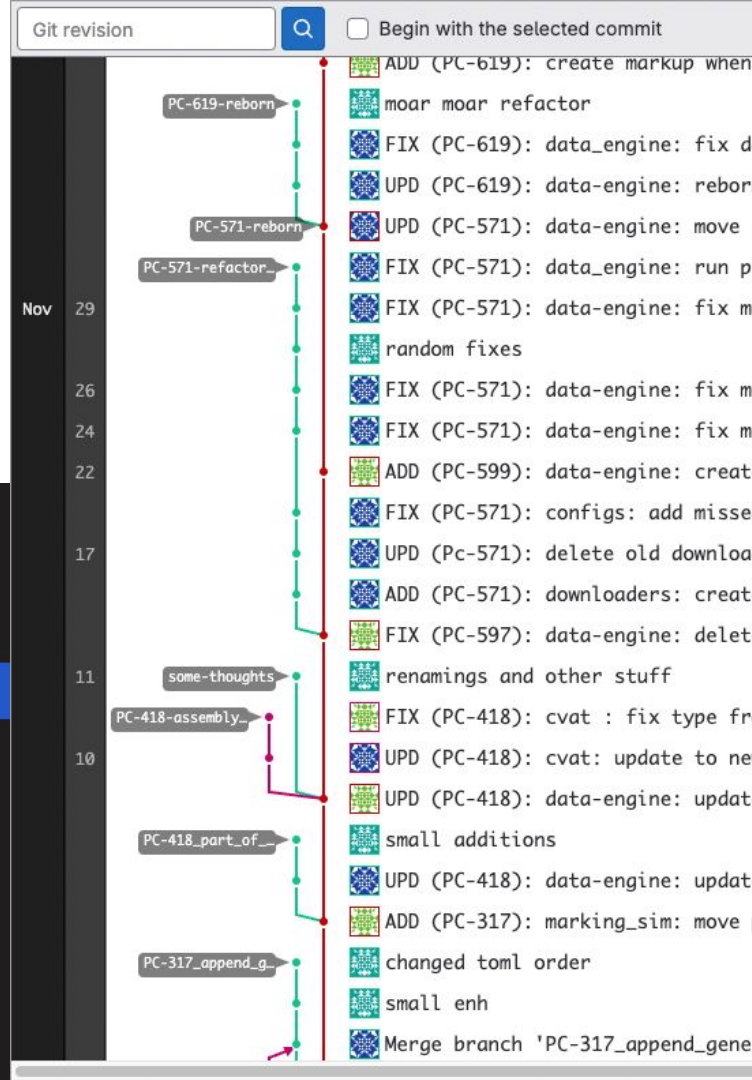- инструметы контроля качества

Хранение кода

girafe
ai

01

Хранение кода

- Одна локальная копия
- Много локальных копий
- Удалённые копии
- Система контроля версий (git, svn, etc.)

Version control systems (VCS)

- Local only
  - Source Code Control System (SCCS) – part of UNIX [1973]
- Client–server
  - Apache Subversion (SVN) [2000]
  - Yandex Arcadia (fork of SVN)

S U B V E R S I O N

Version control systems (VCS)

- Local only
  - Source Code Control System (SCCS) [1973]
- Client–server
  - Apache Subversion (SVN) [2000]
  - Yandex Arcadia (fork of SVN)
- Distributed
  - Git [2005]
  - Mercurial [2005]
  - Plastic (.NET) [2006]
  - Perforce [1995]
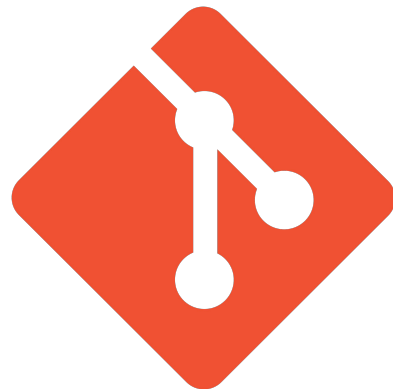
PERFORCE

mercurial

Git servers

- GitHub [2008]
  - proprietary
  - owned by Microsoft
- GitLab [2011]
  - open source, but not free
- Gitea [2016]
  - fork of Gogs FOSS
- Gogs [2014]
  - FOSS
  - example of running service - GIN

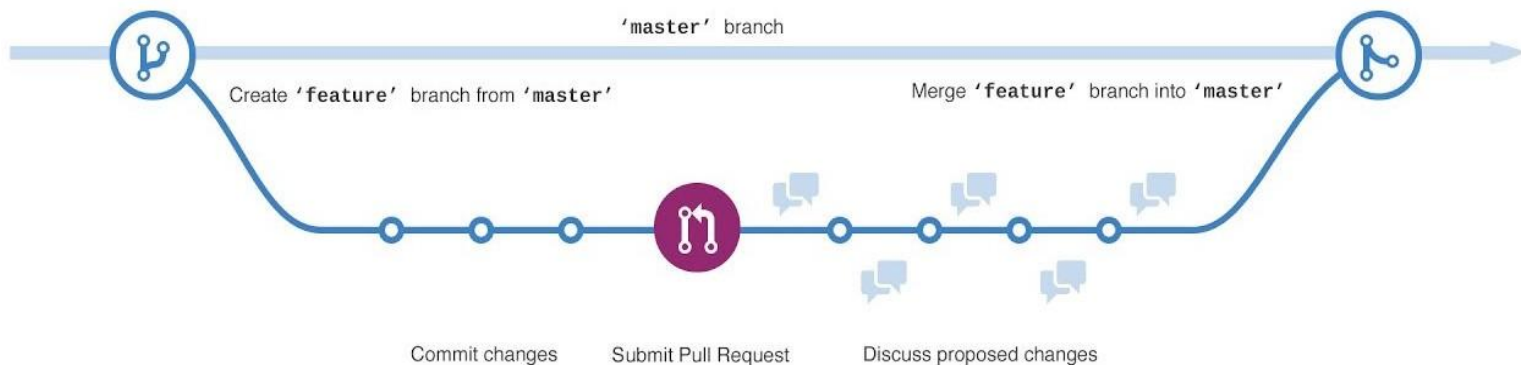https://alternativeto.net/software/github/?license=opensource
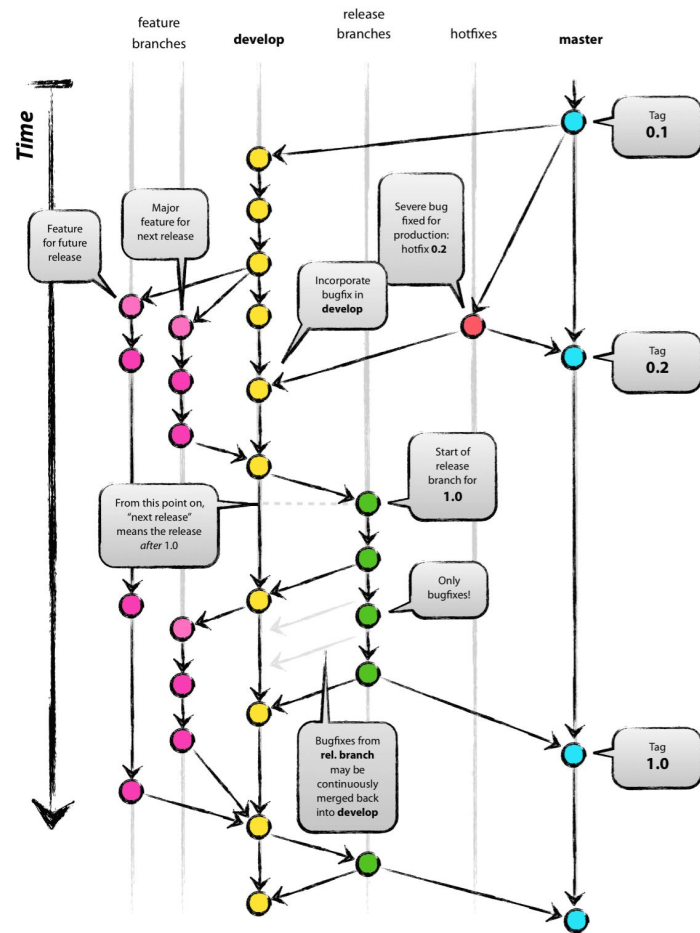
Работа в команде

girafe
ai

02

Работа в команде

- Best practice - [merge requests (pull requests)](#)

Работа в команде

- Best practice - merge requests (pull requests)
  - Advanced - Git Flow (original article)
  - Flows comparison

Git rebase

git Merge

git Rebase

master

feature

master

feature

master

feature
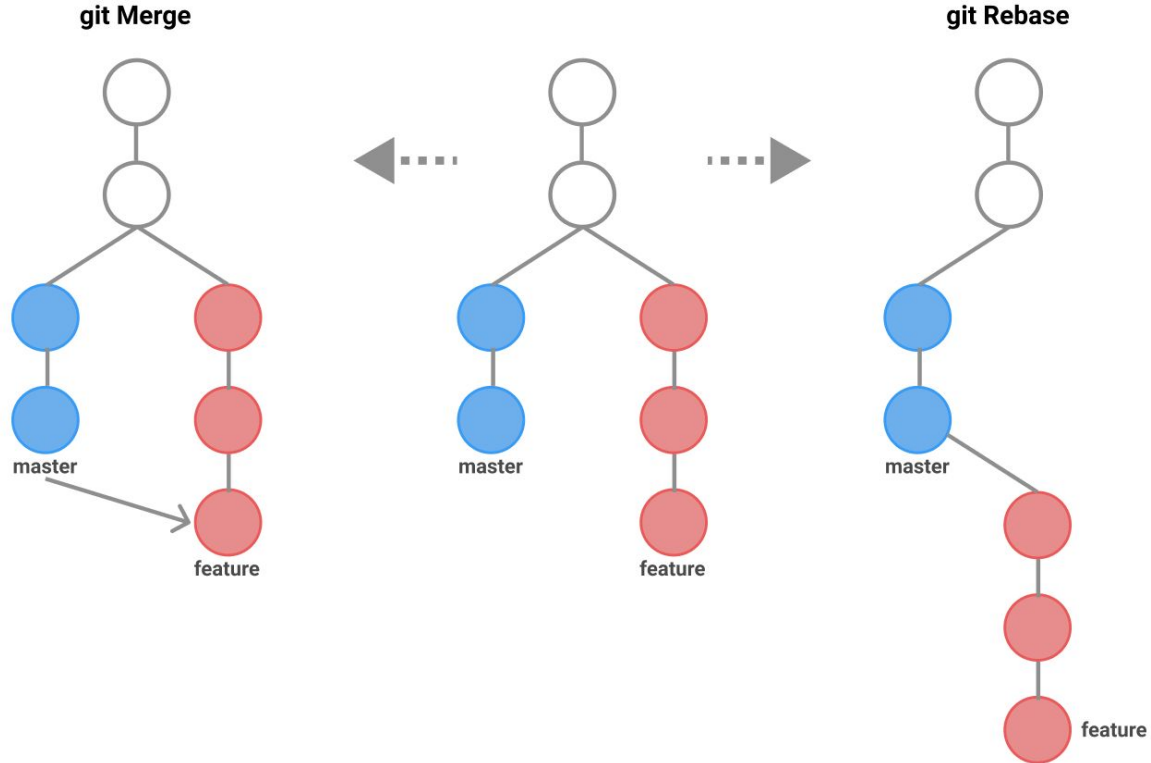
Дистрибуция кода

- Source code
  - клонируем проект на финальную машину
  - обновляем с помощью git pull
  - не требует структуры
- Пакеты
  - устанавливаем с помощью pip install
  - обновляем через pip update
  - предполагает структуру
  - для простого пакетирования подходит poetry
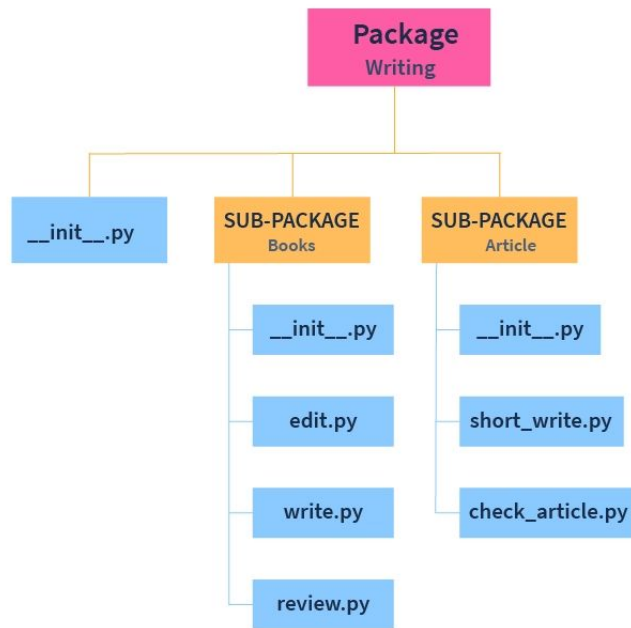
Continuous Integration (CI) & Continuous Delivery (CD)

girafe
ai

03

CI/CD stacks

- Github Actions
- GitLab CI
- Gitea Actions

TEST

COMMIT

UNIT TEST

REVIEW   STAGING   PRODUCTION

CODE

BUILD   INTEGRATION TESTS

RELATED CODE

CI PIPELINE   **CD PIPELINE**

Структура кодовой базы

girafe
ai

04

Стандарты форматирования кода

- PEP 8 (code style), PEP 257 (docstrings)
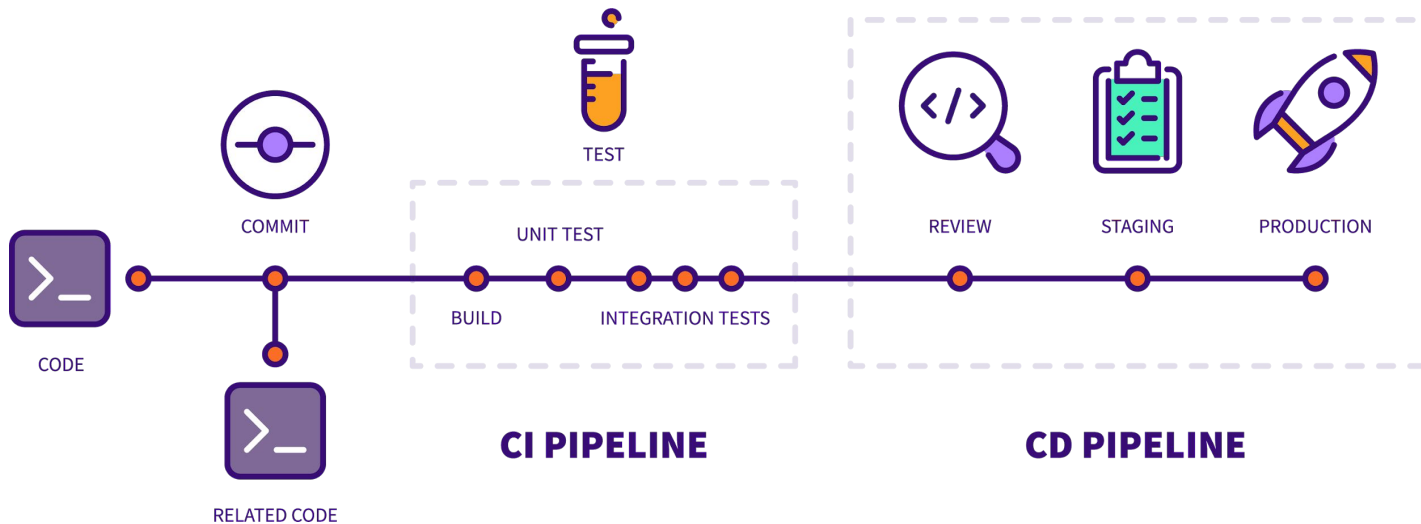- Google Python Style Guide
- Black code style (rationale)

A Foolish Consistency is the Hobgoblin of Little Minds

One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts".

A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is the most important.

However, know when to be inconsistent – sometimes style guide recommendations just aren't applicable. When in doubt, use your best judgment. Look at other examples and decide what looks best. And don't hesitate to ask!

Code Style

- [pre-commit](#)
- [black](#) ([PyCon talk](#))

  The uncompromising code formatter

  Other solutions:
  - yapf
  - autopep8


```python
class Foo  (        object  ):
  def f     (self
  ):
    return         37*-2
  def g(self, x,y=42):
    return y
def f  (    a: List[ int ]) :
  return        37-a[42-u :  y**3]
```

```python
class Foo(object):
    def f(self):
        return 37 * -2

    def g(self, x, y=42):
        return y


def f(a: List[int]):
    return 37 - a[42 - u : y ** 3]
```

Code Style

- pre-commit
- black
- isort

  Configurable

  Sorts import to 3 categories:
  - standard library
  - external (third party)
  - internal (this project and associatives)

isort your imports, so you don't have to.

```
import os
from math import ceil
import myproject.test

import .utils
from math import sqrt
import django.settings
```

```
from math import ceil, sqrt
import os

import django.settings

import myproject.test
```

Code Style

- pre-commit
- black
- isort
- flake8
  List of rules explained

More solutions from Python Code Quality Authority:
  - bandit

Example of rules:

- ❖ F401: Module imported but unused
- ❖ E402: Module level import not at top of file
- ❖ F811: Redefinition of unused name from line n

Code Style

- pre-commit
- black
- isort
- flake8
- mypy

  Best for code editors,

  not pre-commit

: my[py]

Seamless dynamic and static typing

From Python...

```
def fib(n):
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

...to statically typed Python

```
def fib(n: int) -> Iterator[int]:
    a, b = 0, 1
    while a < n:
        yield a
        a, b = b, a+b
```

Code Style

- pre-commit
- black
- isort
- flake8
- mypy
- nbQA

  Same functionality for Jupyter Notebooks!

  Supports pre-commit

nbQA

Quality Assurance For Jupyter Notebooks

Code Style

- pre-commit
- black
- isort
- flake8
- mypy
- nbQA
- prettier

  Formats YAML and Markdown files to common format

Code Style

- pre-commit
- black
- isort
- flake8
- mypy
- nbQA
- prettier

- annotations, typing module
- Documentation Driven Development (docstrings)
- Sphinx
- pathlib

О чём поговорили
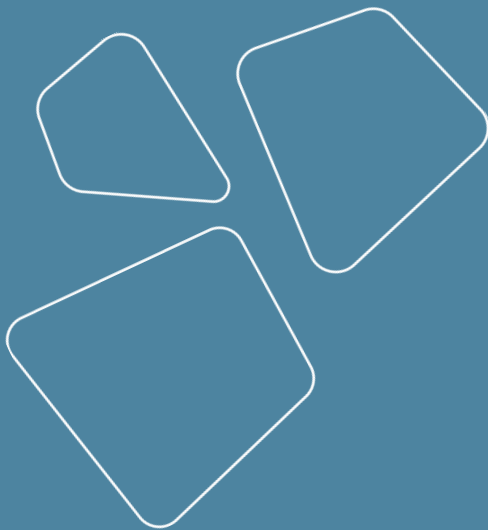
- Хранение кода
- Git servers
- Разработка в больших командах
  - Дистрибуция кода
- CI/CD
- Форматирование кода
- Inline документация

- pre-commit
- инструметы контроля качества

Спасибо за внимание!

Вопросы?