

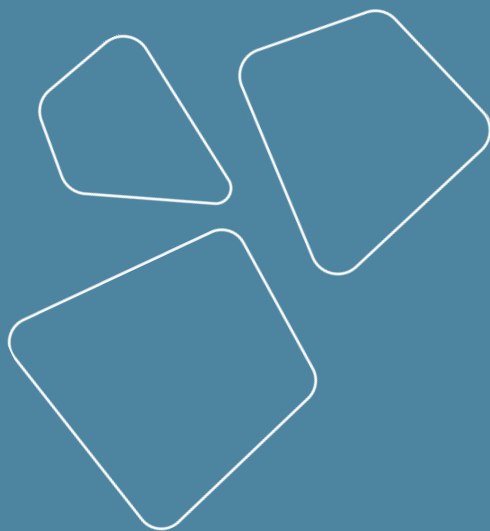
# Управление кодом и `git servers`

Vladislav Goncharenko



HSE, fall 2023

# Ресар

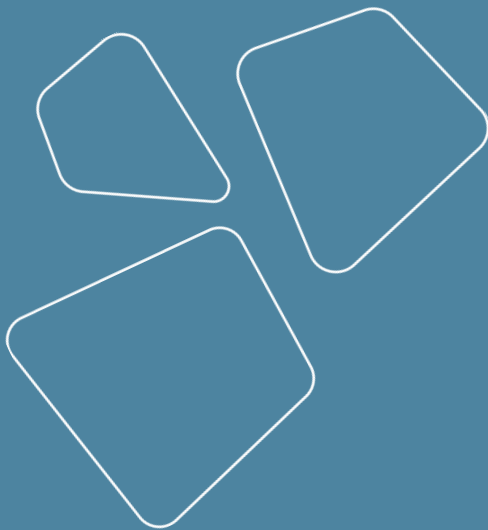


- Зачем нужен MLOps
- Стандарты разработки Data Science проектов
- Что такое MLOps
- Подзадачи MLOps
- Обзор решений от крупных компаний
- Проблемы open source стека

## Practice:

- 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

Practice:

- pre-commit
- инструменты контроля качества

# Хранение кода

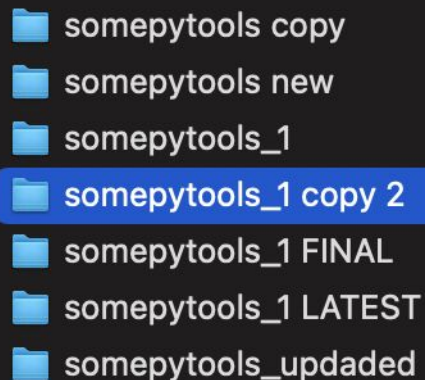
---

girafe  
ai

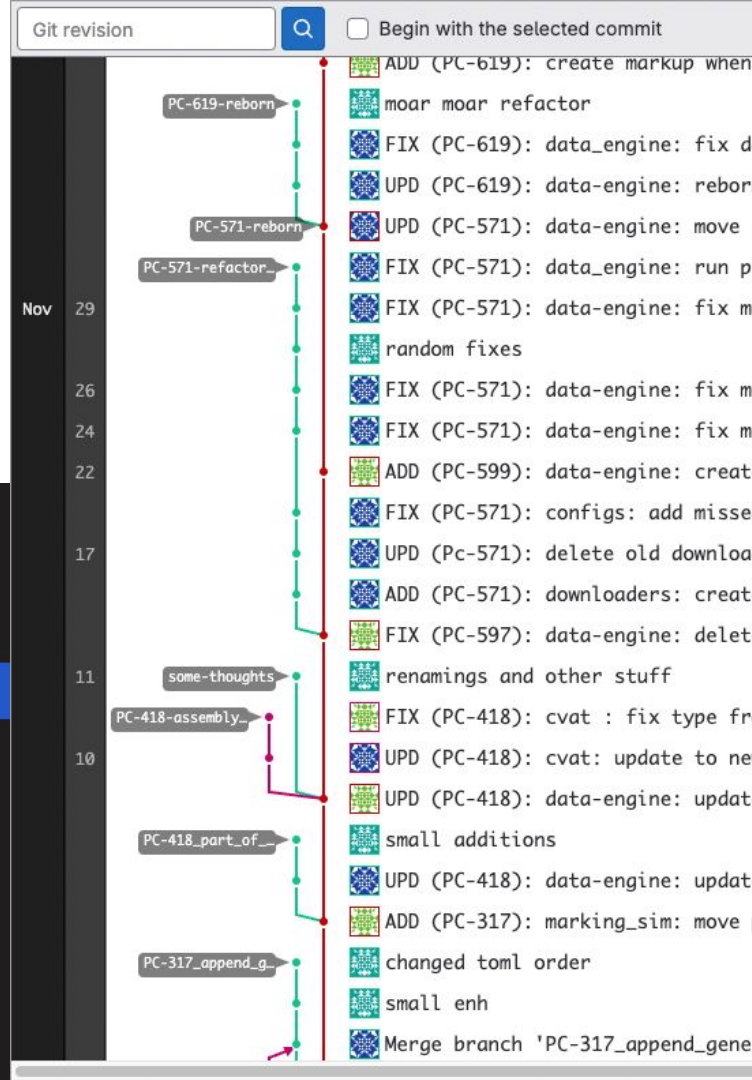
01

# Хранение кода

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



somepytools copy  
somepytools new  
somepytools\_1  
**somepytools\_1 copy 2**  
somepytools\_1 FINAL  
somepytools\_1 LATEST  
somepytools\_updated



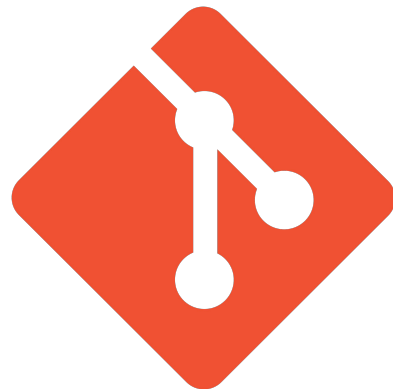
# Version control systems (VCS)

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



# 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

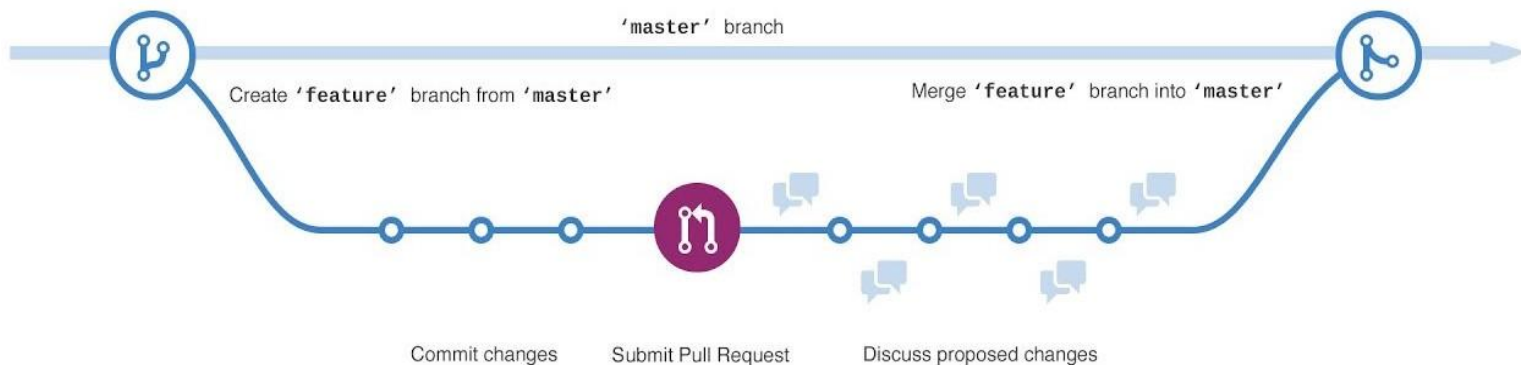
---

girafe  
ai

02

# Работа в команде

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



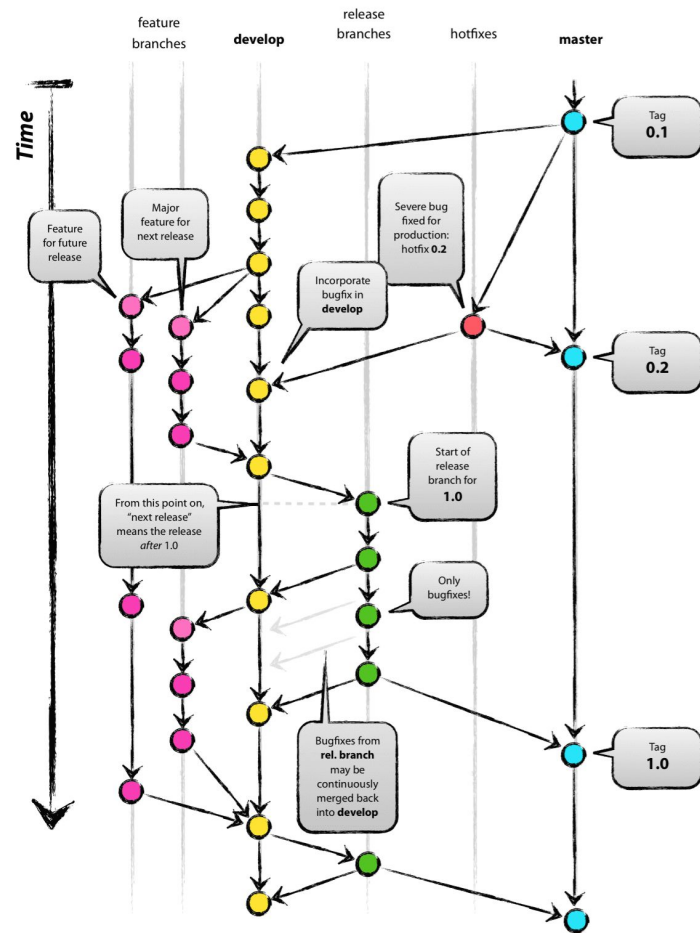
# Работа в команде

- Best practice - [merge requests \(pull requests\)](#)
  - Advanced - [Git Flow \(original article\)](#)
  - [Flows comparison](#)

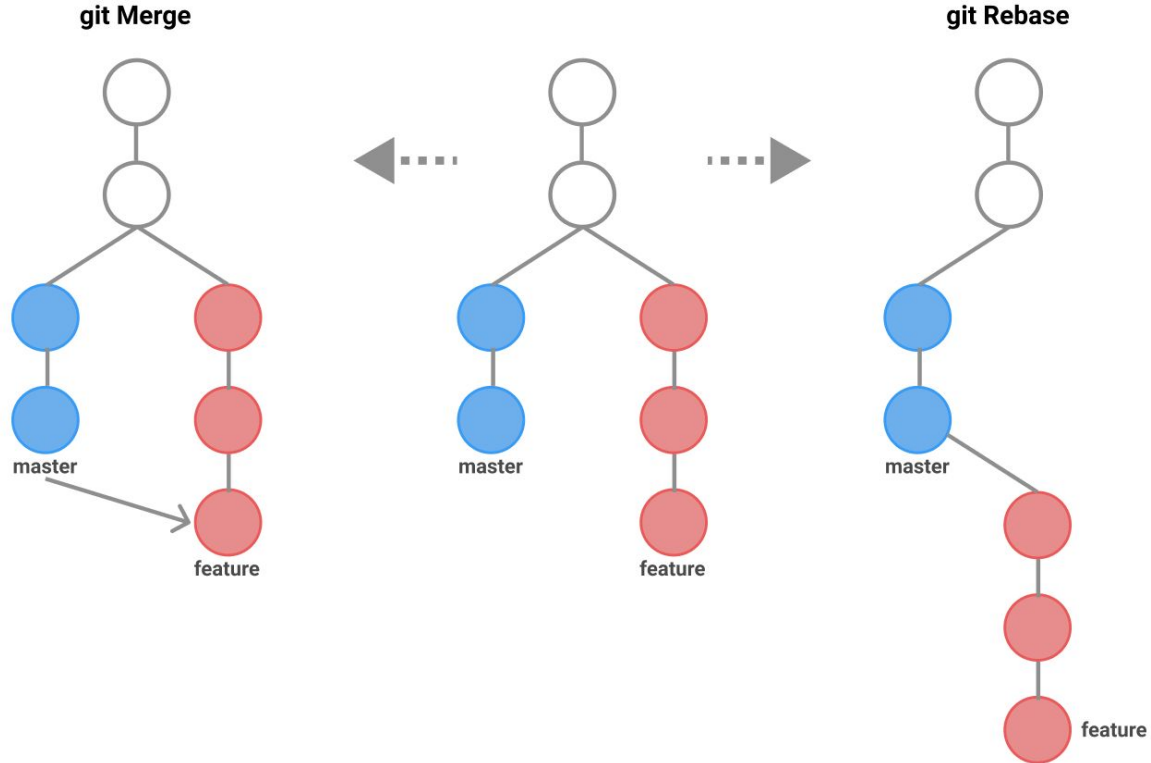
**Brilliant git tutorial by Atlassian:**

<https://www.atlassian.com/git/tutorials/what-is-version-control>

Covers topics from basics to advanced

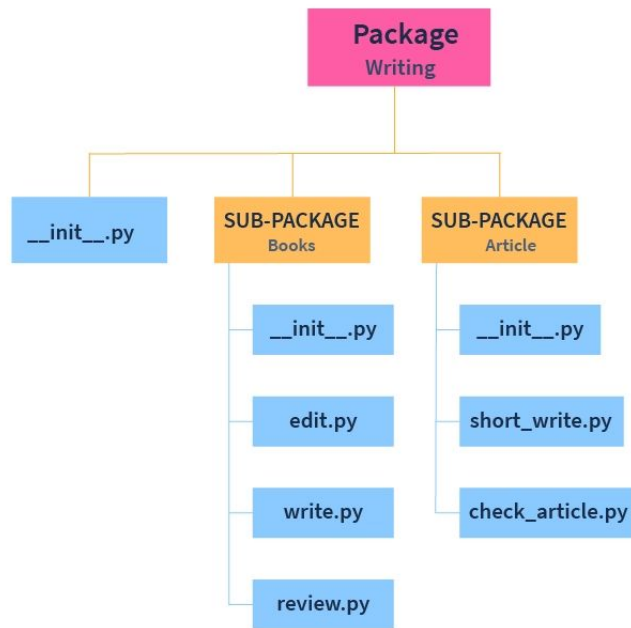


# Git rebase



# 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

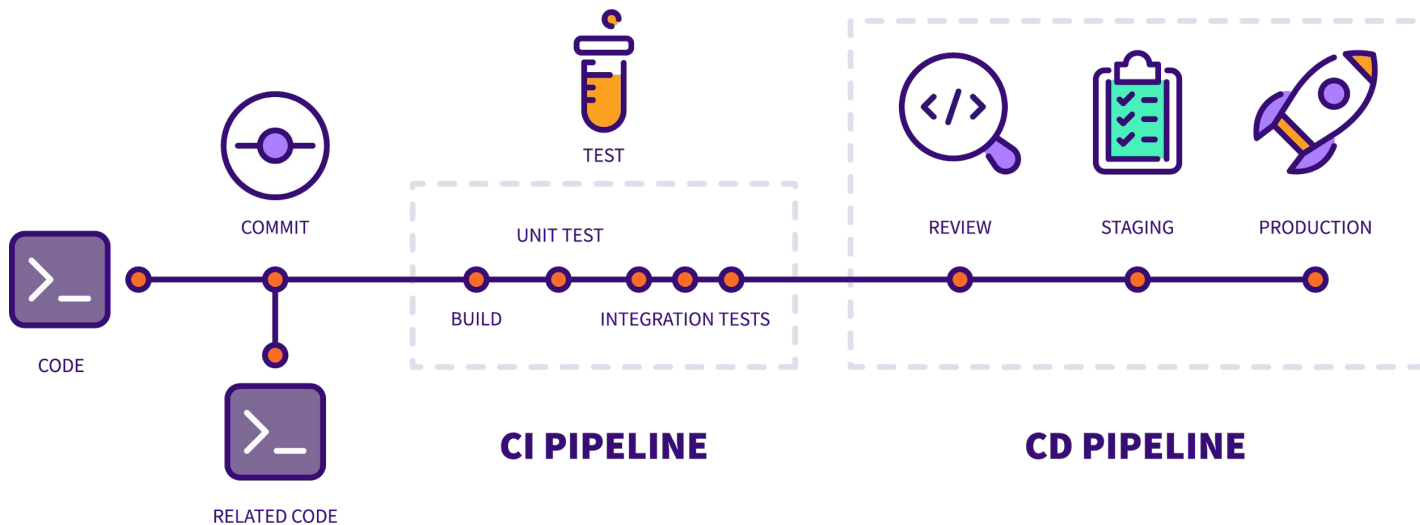
# CI/CD stacks

## Git server integrated

- Github Actions
- GitLab CI
- Gitea Actions

## Standalone

- Jenkins
- Travis
- Circle CI
- TeamCity
- Bamboo
- Circle CI
- TeamCity



# Code quality tools

---

girafe  
ai

04



# Code styles

- Python standards
  - [PEP 8](#) (code style), [PEP 257](#) (docstrings)
- Custom standards
  - [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 formatters

## [Black](#)

The uncompromising code formatter

[PyCon talk](#)

Other solutions:

- [yapf](#) - configurable
- [autopep8](#) - only pep8



```
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]
```

```
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 formatters

- [isort](#)

Configurable

Sorts import to 3

- standard library
- external (third party)
- internal (this project and associatives)



```
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

- [black](#)
- [isort](#)
- [flake8](#)
  - [List of rules explained](#)
  - [pylint](#)

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

More solutions from [Python Code Quality Authority](#):

- [bandit](#)



# Code Style

- [black](#)
- [isort](#)
- [flake8](#)
- [mypy](#)

Best for code editors,  
not pre-commit



## 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

- [black](#)
- [isort](#)
- [flake8](#)
- [mypy](#)
- [nbQA](#)

Same functionality for Jupyter Notebooks!

Supports pre-commit



**nbQA**  
Quality Assurance For Jupyter Notebooks

# Code Style

- [black](#)
- [isort](#)
- [flake8](#)
- [mypy](#)
- [nbQA](#)
- [prettier](#)

Formats YAML and Markdown  
format



# Prettier

# Code Style

- [black](#)
- [isort](#)
- [flake8](#)
- [mypy](#)
- [nbQA](#)
- [prettier](#)
- [annotations](#), [typing module](#)
- [Documentation Driven Development](#) ([docstrings](#))
- [Sphinx](#)
- [pathlib](#)





# Pre-commit

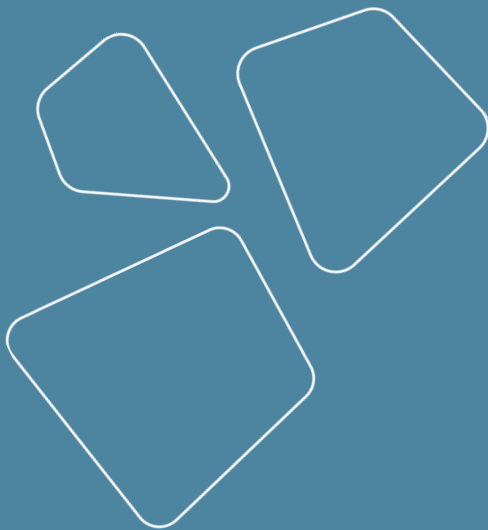
## [pre-commit](#)

uses [git hooks](#) to run common utilities before git commands processing



```
$ pre-commit install
pre-commit installed at /home/asottile/workspace/pytest/.git/hooks/pre-commit
$ git commit -m "Add super awesome feature"
black.....Passed
blacken-docs.....(no files to check)Skipped
Trim Trailing Whitespace.....Passed
Fix End of Files.....Passed
Check Yaml.....(no files to check)Skipped
Debug Statements (Python).....Passed
Flake8.....Passed
Reorder python imports.....Passed
pyupgrade.....Passed
rst ``code`` is two backticks.....(no files to check)Skipped
rst.....(no files to check)Skipped
changelog filenames.....(no files to check)Skipped
[master 146c6c2c] Add super awesome feature
1 file changed, 1 insertion(+)
```

# Revise



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

Practice:

- pre-commit
- инструменты контроля качества

# Спасибо за внимание!

Вопросы?

