# Запуск Git-хуков при помощи pre-commit

*Опубликовано: 31 мая 2020, Вс, автор: Андрей Семакин (https://semakin.dev/author/andrei-semakin.html)*
*Обновлено: 31 мая 2020, Вс*
*8 минут*

# Git-хуки

Умение работать с системой контроля версий Git – базовый навык для выживания разработчика (на любом языке программирования) в современных реалиях. Система контроля версий – это этакая машина времени для вашего проекта: всегда можно вернуться на любое прошлое состояние проекта, понять когда, как, что и кем менялось.

*Интересный факт:* согласно Google Trends (https://trends.google.com/trends/explore?date=all&q=%2Fm%2F05vqwg,%2Fm%2F012ct9,%2Fm%2F08441_,%2Fm%2F08w6d6,%2Fm%2F09d6g&hl=en-US&tz=&tz=), во всём мире Git фактически вытеснил другие системы контроля версий. Только в Китае почему-то до сих пор популярен Subversion (46% рынка). К чему бы это?

Git имеет крайне полезную фичу – возможность исполнять произвольный код на многих этапах работы через так называемые хуки. Вот примеры доступных хуков:

- `pre-commit` – выполняется перед созданием коммита;
- `commit-msg` – выполняется после добавления сообщения коммита;
- `post-checkout` – выполняется после переключения ветки;
- `pre-push` – выполняется перед загрузкой локальной истории на удалённый сервер.

Полный список хуков можно посмотреть в документации (https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks).

Как это работает? Рассмотрим, например, схему работы хука `pre-commit`:

1. пользователь пишет в терминале `git commit -v`;

2. `git` пытается выполнить хук `pre-commit` локально, на машине разработчика;
3. если хук завершается ошибкой, то операция коммита прерывается;
4. если хук выполнился без ошибок, то операция коммита продолжается, открывается текстовый редактор для ввода сообщения.

Фишка в том, что хуки могут прерывать операции, за которые они отвечают, если что-то идёт не так. Это идеальное место, чтобы запускать какие-нибудь проверки качества кода, например, линтеры, форматтеры или тесты (если они работают быстро, конечно), или проверять сообщение коммита на соответствие конвенциям или на грамматические ошибки.

# Пишем Git-хук на `bash`

Давайте для понимания напишем простой скрипт, который будет использоваться в качестве хука.

В этом примере я использую Linux. Если вы пользуетесь Windows, то большинство описанный вещей будут работать без изменений через Git Bash, который устанавливается вместе с Git for Windows. Возможно, вам только придётся поменять путь до вашего `bash` в шебанге (https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D0%B1%D0%B0%D0%BD%D0%B3_(Unix)), как описано в конце этой статьи (https://www.tygertec.com/git-hooks-practical-uses-windows/). Если же вы соберётесь писать в хуках что-то сложное и требующее интеграции с ОС, то, возможно, стоит вместо `bash` использовать PowerShell.

Создадим пустой репозиторий:

```
$ git init git_hooks_example
$ cd git_hooks_example/
```

Git уже заботливо создал для нас шаблоны для написания хуков, которые лежат в специальной служебной директории:

```
$ ls -l .git/hooks/
total 56
-rwxr-xr-x. 1 br0ke br0ke  482 May 30 21:36 applypatch-msg.sample*
-rwxr-xr-x. 1 br0ke br0ke  900 May 30 21:36 commit-msg.sample*
-rwxr-xr-x. 1 br0ke br0ke 4655 May 30 21:36 fsmonitor-watchman.sample*
-rwxr-xr-x. 1 br0ke br0ke  193 May 30 21:36 post-update.sample*
-rwxr-xr-x. 1 br0ke br0ke  428 May 30 21:36 pre-applypatch.sample*
-rwxr-xr-x. 1 br0ke br0ke 1647 May 30 21:36 pre-commit.sample*
-rwxr-xr-x. 1 br0ke br0ke  420 May 30 21:36 pre-merge-commit.sample*
-rwxr-xr-x. 1 br0ke br0ke 1496 May 30 21:36 prepare-commit-msg.sample*
-rwxr-xr-x. 1 br0ke br0ke 1352 May 30 21:36 pre-push.sample*
-rwxr-xr-x. 1 br0ke br0ke 4902 May 30 21:36 pre-rebase.sample*
-rwxr-xr-x. 1 br0ke br0ke  548 May 30 21:36 pre-receive.sample*
-rwxr-xr-x. 1 br0ke br0ke 3639 May 30 21:36 update.sample*
```

Можно, например, просто переименовать файл `pre-commit.sample` в `pre-commit`, и этот хук вступит в силу. В моём случае там на `bash` реализована проверка имён файлов, которая не допустит добавление файлов с именами, содержащими не-ASCII символы.

Давайте не будем этого делать, а взамен напишем простой скрипт, который будет запускать все нужные нам линтеры (https://semakin.dev/2020/05/python_linters/) и форматтеры (https://semakin.dev/2020/05/black/):

```
#!/usr/bin/env bash

# Получаем список файлов, которые пользователь пытается закоммитить,
# и выбираем из них те, которые заканчиваются на `.py`.
# Взято отсюда: https://stackoverflow.com/a/3068990/10650942.
CHANGED_PYTHON_FILES=$(git diff --cached --name-only --diff-filter=ACMR
| grep ".py\$")

if [ -z "$CHANGED_PYTHON_FILES" ]
then
   echo "No Python files found. No reason to run checks."
   exit 0
fi

# Включаем режим, в котором любая ошибка сразу же завершит весь скрипт
ошибкой.
set -e

# Запускаем проверки.
# Если хотя бы одна завершится ошибкой, то операция будет прервана.
flake8 $CHANGED_PYTHON_FILES
black --check $CHANGED_PYTHON_FILES

echo "All checks successfully passed."
```

Этот файл должен быть сохранён по пути `.git/hooks/pre-commit`. Файл нужно сделать исполняемым, иначе Git не сможет его запустить:

```
$ chmod a+x .git/hooks/pre-commit
```

Теперь давайте создадим какой-нибудь файл, который точно не пройдет этих проверок. Обратите внимание на лишние пробелы рядом со скобками, которые не понравятся `flake8`, и на одинарные кавычки, которые не понравятся `black`:

```
print( 'Hello world!' )
```

Попытаемся его закоммитить:

```
$ git add example.py
$ git commit -v
example.py:1:7: E201 whitespace after '('
example.py:1:22: E202 whitespace before ')'
```

Наш умный хук выполнился и запустил `flake8`, который нашел ошибки. Давайте удалим лишние пробелы и попытаемся закоммитить ещё раз:

```
$ git add example.py
$ git commit -v
would reformat example.py
Oh no! 💥 💔 💥
1 file would be reformatted.
```

В этот раз `flake8` не нашёл ошибок, но операция всё равно завершилась ошибкой из-за `black`. Давайте отформатируем файл и наконец закоммитим его:

```
$ black .
$ git add example.py
$ git commit -v
```

На этот раз всё срабатывает успешно.

Чтобы закоммитить, игнорируя хуки, можно было передать в команду флаг `--no-verify` или `-n`:

```
git commit -v -n
```

Как видите, писать git-хуки самостоятельно не так уж и сложно. Не обязательно использовать `bash`, хуки можно писать на чём угодно — любой исполняемый файл подойдет, хоть на интерпретируемом языке, хоть на компилируемом. Если бы нужно было писать какую-то более сложную логику, то я бы, например, предпочёл написать хук на Python.

У такого ручного способа есть несколько недостатков:

- хук хранится в директории `.git`, которая является служебной для локальной копии репозитория и сама не сохраняется в систему контроля версий; это значит, что если вы захотите поделиться своим хуком с коллегами, то вы можете лишь каким-либо образом скинуть им исходный код, а им придётся самостоятельно класть его в правильное место и назначать права;
- первая команда, завершившаяся ошибкой, прерывает выполнение хука и последующие команды не выполняются; это можно исправить, но получится сильно сложнее;
- логика хука может быстро стать сложной;
- всё нужно имплементить самостоятельно;
- наш хук полагается на наличие команд `flake8` и `black`, так что они либо должны быть установлены глобально, либо перед коммитом нужно активировать виртуальное окружение проекта.

# Используем готовый инструмент — `pre-commit`

Как обычно, для всего уже есть решения. Представляю вашему вниманию pre-commit (https://pre-commit.com/) — умный инструмент для управления Git-хуками.

## Установка

`pre-commit` написан на Python (ещё бы, иначе б я не был его фанатом 😁), поэтому установить его можно через `pip`. Он должен быть установлен глобально, а не в виртуальном окружении проекта, где вы собираетесь его использовать. Рекомендую использовать метод установки в домашнюю директорию пользователя (см. заметку про виртуальные окружения (https://semakin.dev/2020/04/python_virtualenv/) за подробностями):

```
$ pip install --user pre-commit
```

Проверим установку:

```
$ pre-commit --version
pre-commit 2.4.0
```

# Настройка

`pre-commit` спроектирован с прицелом на удобное использование сторонних переиспользуемых хуков, но может исполнять и вообще любые команды. Уже написаны сотни полезных хуков (https://pre-commit.com/hooks.html) для разных языков и задач, из которых, как из конструктора, можно собрать практически любой нужный вам вокрфлоу. Выбирайте те, которые вам понравятся. Для примера я возьму всё те же `flake8` и `black`, и ещё несколько других хуков сверху (а что, бесплатно же).

`pre-commit` конфигурируется на уровне репозитория при помощи YAML-файла (https://ru.wikipedia.org/wiki/YAML). Файл должен называться `.pre-commit-config.yaml` и находиться в корне репозитория. Давайте сгенерируем базовый конфиг:

```
$ pre-commit sample-config > .pre-commit-config.yaml
```

И допилим примерно до такого состояния:

```
# See https://pre-commit.com for more information
# See https://pre-commit.com/hooks.html for more hooks
repos:
  - repo: 'https://github.com/pre-commit/pre-commit-hooks'
    rev: v2.4.0
    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: check-byte-order-marker
      # предотвращает сохранение приватных ключей
      - id: detect-private-key
      # проверяет, что файлы, которые мы собираемся сохранять, как мини
мум валидный Python
      - id: check-ast
  - repo: 'https://gitlab.com/pycqa/flake8'
    rev: 3.8.2
    hooks:
      - id: flake8
  - repo: 'https://github.com/psf/black'
    rev: 19.10b0
    hooks:
      - id: black
```

Теперь включим `pre-commit` в текущем репозитории.

```
$ pre-commit install
```

Рекомендую добавить эту команду в документацию для разработчиков. Это всё, что нужно будет сделать вашим коллегам, чтобы хуки заработали и у них тоже.

Можно убедиться, что `pre-commit` заменил наш старый хук на свой:

```
$ cat .git/hooks/pre-commit
#!/usr/bin/env python3.8
# File generated by pre-commit: https://pre-commit.com
...
```

При этом старый хук был переименован в `pre-commit.legacy`. Умный `pre-commit` будет запускать и его тоже, чтобы не сломать текущее поведение. Если это не желательно, то проще всего удалить этот старый файл:

```
$ rm .git/hooks/pre-commit.legacy
```

Проверим, что конфигурационный файл валиден, а заодно и что всё нынешнее содержимое репозитория удовлетворяет описанными правилам:

```
$ pre-commit run --all-files
```

При первом запуске `pre-commit` скачает и закэширует все файлы, необходимые для выполнения проверок. Для проверок, которые написаны на Python, будут созданы изолированные виртуальные окружения, так что они никак не будут зависеть от виртуального окружения проекта. Первый раз из-за скачивания зависимостей эта команда может выполняться секунд 30 (в зависимости от скорости интернета), но перезапустите её ещё раз и она завершится за секунду.

Возможно, `pre-commit` найдёт проблемы или даже исправит некоторые файлы.

Если проверка отработала без ошибок, то конфиг нужно добавить в Git:

```
$ git add .pre-commit-config.yaml
$ git commit -m “Add pre-commit configuration”
```

# Плагины/зависимости для проверок

Из-за того, что для проверок, написанных на Python, создаются отдельные виртуальные окружения, может быть не совсем понятно, как устанавливать плагины для таких программ, как, например, `flake8`. Для `flake8` важно, чтобы все его плагины были установлены с ним в одно виртуальное окружение, иначе он просто не сможет их найти. Специально для этого у `pre-commit` предусмотрена настройка `additional_dependencies`, которая используется вот таким образом:

```
  - repo: 'https://gitlab.com/pycqa/flake8'
    rev: 3.8.2
    hooks:
      - id: flake8
        additional_dependencies:
          - flake8-bugbear
```

При следующем запуске `pre-commit` обнаружит новую зависимость и установит её в то же виртуальное окружение, что и `flake8`. Теперь перед коммитом будет выполняться не просто голый `flake8`, но ещё и с дополнительным плагином. Таких зависимостей может быть сколько угодно.

# Использование

Теперь можно вообще забыть про существование `pre-commit` и просто пользоваться Git как обычно, а `pre-commit` будет выполнять все описанные проверки, изредка беспокоя вас прерванными операциями, если будут найдены какие-нибудь проблемы. Давайте снова попробуем закоммитить тот сломанный файл с пробелами и кавычками:

```
$ git add example.py
$ git commit -v
Fix End of Files.....................................Passed
Check for added large files.....................Passed
Check for byte-order marker.....................Passed
Detect Private Key.................................Passed
Check python ast....................................Passed
flake8..............................................Failed
- hook id: flake8
- exit code: 1
example.py:1:7: E201 whitespace after '('
example.py:1:22: E202 whitespace before ')'
black...............................................Failed
- hook id: black
- files were modified by this hook
reformatted example.py
All done! ✨ 🍰 ✨
1 file reformatted.
```

Ожидаемо, операция прервалась, но в этот раз мы получили все нужные сообщения об ошибках. Кроме того, файл даже был автоматически отформатирован при помощи `black`, так что ничего даже вручную делать не нужно. Просто ещё раз запускаем те же команды, и в этот раз проверки должны пройти.

На YAML программировать намного приятнее, чем на `bash`! Да, честно говоря, практически на чём угодно писать приятнее, чем на `bash`.

# Альтернативы `pre-commit`

Мне проще всего использовать инструменты, написанные на знакомом мне языке, но вообще `pre-commit` далеко не уникальный инструмент и имеет множество альтернатив. Если вы пишете не на Python, то может быть вам будут ближе другие инструменты, хотя все они имеют примерно схожий функционал:

- `husky` на JS (https://github.com/typicode/husky);
- `lefthook` на Go (https://github.com/Arkweid/lefthook);
- `overcommit` на Ruby (https://github.com/sds/overcommit).

Возможно, вы также найдете для себя что-то полезное на странице “Awesome Git hooks” (https://github.com/aitemr/awesome-git-hooks).

# Заключение

Я всегда стараюсь использовать Git-хуки для запуска всех быстрых проверок. Это не дает мне забывать о проверках, и позволяет быстрее получать обратную связь.

Представьте ситуацию, когда сидишь и ждёшь результатов проверок от CI, которые могут быть достаточно долгими (на проекте, где я сейчас работаю, тесты выполняются 8-10 минут), видишь красный крестик, идёшь посмотреть, что же там сломалось, а там всё *почти* отлично — тесты прошли, но только `flake8` нашёл лишний пробел. Чинишь лишний пробел и снова ждёшь 10 минут, чтобы получить свою зелёную галочку. Дак вот хуки спасают от таких ситуаций, потому что все тривиальные проблемы обнаруживаются за несколько секунд локально на моей машине и никогда не попадают в историю Git.

Настоятельно рекомендую пользоваться Git-хуками. Это позволит вам не тратить время на ерунду, и в итоге быть более эффективным и довольным разработчиком.

Примеры из поста можно найти здесь (https://github.com/and-semakin/git_hooks_example).

> *Если понравилась статья, то подпишитесь на уведомления (https://semakin.dev/pages/subscribe/) о новых постах в блоге, чтобы ничего не пропустить!*

# Ссылки

- документация `pre-commit` (https://pre-commit.com/);
- реестр хуков для `pre-commit` (https://pre-commit.com/hooks.html);
- репозиторий `pre-commit` на GitHub (https://github.com/pre-commit/pre-commit);
- документация Git про хуки (https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks).

тэги: python (https://semakin.dev/tag/python.html), pre-commit (https://semakin.dev/tag/pre-commit.html), git (https://semakin.dev/tag/git.html)

---

2 Comments - *powered by utteranc.es*

Arsennikum commented on 1 июн. 2020 г.

yaml-developer :D

а как он работает? Когда запускаешь pre-commit install он просто, получается, генерит .git/hooks/pre-commit на баше?
Но тогда при любом изменении .pre-commit-config.yaml нужно снова запускать pre-commit install, получается?

❤️ 1

and-semakin commented on 1 июн. 2020 г. Owner

Не совсем так. В `.git/hooks/pre-commit` складывается по сути статичный скрипт на Python, который не зависит от YAML-файла. Он просто запускает `pre-commit` с нужными аргументами. При редактировании YAML-конфига ничего заново устанавливать не нужно.

👍 1

Write Preview

Sign in to comment

Styling with Markdown is supported Sign in with GitHub

● (/feeds/all.atom.xml) ● (https://t.me/pythonic_attacks)