

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ №4
«Основы git»

Практическая работа
по дисциплине «Современные технологии программирования»
студента 1 курса группы ПИ-б-о-231(2)
Аметов Кемран Ленверович
направления подготовки 09.03.04 «Программная инженерия»

Симферополь, 2024

Цель:

ознакомиться с базовыми возможностями утилиты git, сервисов GitHub, GitLab и возможностей CI/CD, которые они предоставляют.

Ход выполнения задания.

Часть I: Основы Git. GitHub Actions

1. Если у вас ещё не установлена утилита git, то установите её.

Уже установлен

2. Если вы никогда ранее не использовали git, для начала вам необходимо выполните следующие команды, чтобы git узнал ваше имя и электронную почту:

```
git config --global user.name "Your Name"
git config --global user.email "your_email@whatever.com"
```

Уже настроен

3. В git используется три уровня конфигов:

Системный --system - содержит значения, общие для всех пользователей системы и для всех их репозиториях;

Глобальный --global - хранит настройки для всех репозиториях текущего пользователя;

Локальный --local - хранит настройки для конкретного репозитория.

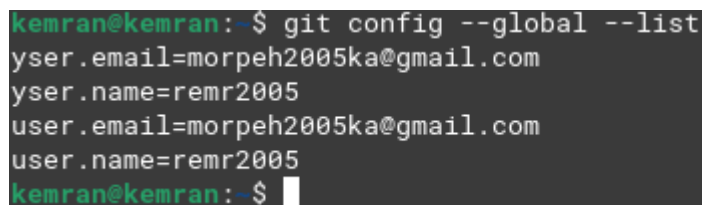
При совпадении настроек применяются более локальные.

4. Введите команду:

```
git config --global --list
```

Вы должны увидеть своё имя пользователя и почту которую указывали ранее.

Ключ --list (-l) отображает список настроек указанного уровня или общие настройки, если уровень не выбран.



```
kemran@kemran:~$ git config --global --list
user.email=morpeh2005ka@gmail.com
user.name=remr2005
user.email=morpeh2005ka@gmail.com
user.name=remr2005
kemran@kemran:~$
```

5. В домашнем каталоге создайте папку "var_keeper".
6. Перейдите в "var_keeper" и создайте в нём новый репозиторий командой: `git init`
7. Убедитесь, что в каталоге "var_keeper" появился каталог ".git" при помощи команды: `ls -al`. Этот каталог и есть репозиторий. В большинстве случаев вам не придётся работать с ним на прямую, вся работа с репозиторием происходит через утилиту git
8. Введите команду:

git config --list --show-originВ данном случае мы используем ключ --list чтобы посмотреть все настройки и ключ --show-origin чтобы посмотреть в каком конфиг-файле эта настройка указана.

9. Введите команду:

```
git status
```

Гит должен сообщить, что вы находитесь в ветке "master" и ещё нет ни одного коммита и коммитить нечего.

Само слово *коммит* - переводят на русский как *фиксация*. По сути коммит является своего рода точкой сохранения. Если в какой-то момент вы создали коммит, то в любое время сможете вернуть все файлы в каталоге к их состоянию на момент создания коммита.

```
kemran@kemran:~$ mkdir var_keeper
kemran@kemran:~$ cd var_keeper/
kemran@kemran:~/var_keeper$ git init
подсказка: Using 'master' as the name for the initial branch. This default branch name
подсказка: is subject to change. To configure the initial branch name to use in all
подсказка: of your new repositories, which will suppress this warning, call:
подсказка:
подсказка:     git config --global init.defaultBranch <name>
подсказка:
подсказка: Names commonly chosen instead of 'master' are 'main', 'trunk' and
подсказка: 'development'. The just-created branch can be renamed via this command:
подсказка:
подсказка:     git branch -m <name>
Инициализирован пустой репозиторий Git в /home/kemran/var_keeper/.git/
kemran@kemran:~/var_keeper$ ls -al
итого 12
drwxrwxr-x   3 kemran kemran 4096 мая 10 22:03 .
drwxr-x---+ 59 kemran kemran 4096 мая 10 22:03 ..
drwxrwxr-x   7 kemran kemran 4096 мая 10 22:03 .git
kemran@kemran:~/var_keeper$ git config --list --show-origin
file:/home/kemran/.gitconfig yser.email=morpeh2005ka@gmail.com
file:/home/kemran/.gitconfig yser.name=remr2005
file:/home/kemran/.gitconfig user.email=morpeh2005ka@gmail.com
file:/home/kemran/.gitconfig user.name=remr2005
file:./git/config core.repositoryformatversion=0
file:./git/config core.filemode=true
file:./git/config core.bare=false
file:./git/config core.logallrefupdates=true
kemran@kemran:~/var_keeper$ git status
Текущая ветка: master

Еще нет коммитов

нечего коммитить (создайте/скопируйте файлы, затем запустите
«git add», чтобы отслеживать их)
kemran@kemran:~/var_keeper$
```

10. Создайте файл "ReadMe.md". В этом файле будет описание проекта и другая полезная для пользователей информация. Пока-что в файле должно быть:

```
# Var_keeper
```

Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.

11. Снова введите команду:

```
git status
```

Гит должен сообщить, что вы по прежнему находитесь в ветке "master", коммитов нет, но есть неотслеживаемый файл "ReadMe.md".

Неотслеживаемые файлы находятся только в рабочем каталоге и после их удаления, средствами git их не восстановить.

12. Введите команды:

```
git add ReadMe.md
git status
```

Теперь неотслеживаемых файлов больше нет, а файл "ReadMe.md" *скопирован* в специальную зону внутри каталога ".git" которая называется "stage" (это своего рода кэш). Т.к. файл именно скопирован, то при изменении "ReadMe.md", в рабочем каталоге, у вас будет две версии - одна в кэше, вторая в рабочем каталоге.

```
kemran@kemran:~/var_keeper$ echo "# Var_keeper
Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.">ReadMe.md
kemran@kemran:~/var_keeper$ git status
Текущая ветка: master

Еще нет коммитов

Неотслеживаемые файлы:
(используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
    ReadMe.md

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
kemran@kemran:~/var_keeper$ git add ReadMe.md
kemran@kemran:~/var_keeper$ git status
Текущая ветка: master

Еще нет коммитов

Изменения, которые будут включены в коммит:
(используйте «git rm --cached <файл>...», чтобы убрать из индекса)
    новый файл:   ReadMe.md

kemran@kemran:~/var_keeper$
```

13. Измените "ReadMe.md" на:

```
# Var_keeper
```

и снова введите `git status`.

Гит сообщит, что файл "ReadMe.md" был изменен и он отличается от того, что находится в кэше.

14. Если мы хотим обновить файл в кэше, то нужно повторно выполнить `git add ReadMe.md`, но мы поступим наоборот, заменим файл в рабочем каталоге на тот, который был закэширован.

Введите команду:

```
git restore ReadMe.md
```

и проверьте, что "ReadMe.md" вернулся в прежнее состояние.

15. Stage зона это ещё не окончательный коммит, а как бы его заготовка. Всё что там находится, при следующей команде `commit` будет добавлено в дерево коммитов.

16. Введите команду:

```
git commit -m "Добавлен ReadMe.md"
```

Ключ `-m` позволяет указать комментарий к коммиту прямо в командной строке.

Примечание: если не указать ключ `-m`, то будет открыт текстовый редактор, в котором нужно будет написать комментарий. После сохранения файла и выхода из редактора будет выполнен коммит. Обычно такой способ применяется, если нужно написать большое пояснение к коммиту.

17. Введите команду:

```
git status
```

Гит должен сообщить, что коммитить нечего и содержимое рабочего каталога соответствует последнему коммиту.

```
kemran@kemran:~/var_keeper$ echo "# Var_keeper"> ReadMe.md
kemran@kemran:~/var_keeper$ git restore ReadMe.md
kemran@kemran:~/var_keeper$ cat ReadMe.md
# Var_keeper

Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.
kemran@kemran:~/var_keeper$ git commit -m "Добавлен ReadMe.md"
[master (корневой коммит) 00f763a] Добавлен ReadMe.md
1 file changed, 3 insertions(+)
create mode 100644 ReadMe.md
kemran@kemran:~/var_keeper$ git status
Текущая ветка: master
нечего коммитить, нет изменений в рабочем каталоге
kemran@kemran:~/var_keeper$
```

18. Прежде, чем перейти непосредственно к разработке создадим специальный файл предназначенный, для того, чтобы автоматически отфильтровывать "мусор" который не должен попадать в репозиторий (например временные файлы и т.п.). Гит просто не будет обращать внимание на файлы и каталоги перечисленные в этом файле.

19. Создайте файл `".gitignore"` в корне репозитория. Если вы работаете в Windows, то можете получить сообщение об ошибке, т.к. имя файла начинается с точки. При создании файла через терминал проблем не будет.

Внимание! У файла нет расширения, он начинается с точки и все символы маленькие.

20. Чтобы не придумывать содержимое файла самому, перейдите на [сайт](#) (на GitHub тоже можно найти репозитории с готовыми `.gitignore`). В поле ввода напишите `"python"` и `"flask"`, а затем нажмите `"Сгенерировать"`. Содержимое страницы скопируйте в `".gitignore"`.

21. Выполните:

```
git add .gitignore
git commit -m "Добавлен .gitignore"
```

22. Проверьте содержимое рабочего каталога (`ls -al`). По умолчанию, файлы и каталоги начинающиеся с точки не отображаются, поэтому без ключа `-a` в списке файлов вы их не увидите.

23. Введите команду:

```
git log
```

Вы должны увидеть информацию о всех коммитах, которые есть в репозитории с указанием хеша коммита, автора, даты создания и комментария к коммиту. Когда коммитов становится много, их удобнее смотреть в более компактном виде. Один из предустановленных вариантов:

```
git log --oneline
```

```
kemran@kemran:~/var_keeper$ touch .gitignore
kemran@kemran:~/var_keeper$ nano .gitignore
kemran@kemran:~/var_keeper$ git add .gitignore
git commit -m "Добавлен .gitignore"
[master a0c9b30] Добавлен .gitignore
1 file changed, 262 insertions(+)
create mode 100644 .gitignore
kemran@kemran:~/var_keeper$ ls -al
итого 24
drwxrwxr-x  3 kemran kemran 4096 мая 10 22:21 .
drwxr-x---+ 59 kemran kemran 4096 мая 10 22:04 ..
drwxrwxr-x  8 kemran kemran 4096 мая 10 22:21 .git
-rw-rw-r--  1 kemran kemran 5813 мая 10 22:21 .gitignore
-rw-rw-r--  1 kemran kemran  190 мая 10 22:16 ReadMe.md
kemran@kemran:~/var_keeper$ git log
commit a0c9b3032a5461291e012b3a260e400b5d4c025e (HEAD -> master)
Author: remr2005 <morpeh2005ka@gmail.com>
Date:   Fri May 10 22:21:46 2024 +0300

    Добавлен .gitignore

commit 00f763ad86bd75847c7390389cc40356cc1970e0
Author: remr2005 <morpeh2005ka@gmail.com>
Date:   Fri May 10 22:17:21 2024 +0300

    Добавлен ReadMe.md
```

24. Для следующего шага вам понадобится учётная запись на github.com. Если ещё не зарегистрировались, то сделайте это.

Уже был зареган

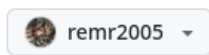
25. На GitHub создайте новый публичный репозиторий с именем `var_keeper`.

Дополнительных галочек выставлять не нужно, он должен быть пустой.

Вообще говоря, имя может не совпадать с именем локального репозитория.

Required fields are marked with an asterisk (*).

Owner *



Repository name *

/ var_keeper

✓ var_keeper is available.

Great repository names are short and memorable. Need inspiration? How about [studious-waddle](#) ?

Description (optional)



Some visibility options are unavailable:

- We are unable to provide access to GitHub private repository services. It appears your account may be based in a U.S.-sanctioned region. As a result, we are unable to provide private repository services and paid services for your account. GitHub has preserved, however, your access to [certain free services for public repositories](#). If your account has been flagged in error, and you are not located in or resident in a sanctioned region, please [file an appeal](#). Please read about [GitHub and Trade Controls](#) for more information.



Public

Anyone on the internet can see this repository. You choose who can commit.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs](#).

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. [Learn more about ignoring files](#).

Choose a license

License: None

A license tells others what they can and can't do with your code. [Learn more about licenses](#).



You are creating a public repository in your personal account.

Create repository

26. После того, как репозиторий будет создан, GitHub покажет шпаргалку, по тому, как можно подружить ваш локальный репозиторий и удалённый. Доступ к репозиторию на GitHub можно настроить по SSH и по HTTPS. Можете выбрать удобный для вас способ. Далее описывается настройка доступа по HTTPS, т.к. он требует меньше дополнительных действий.

У меня уже было настроено ssh соединение

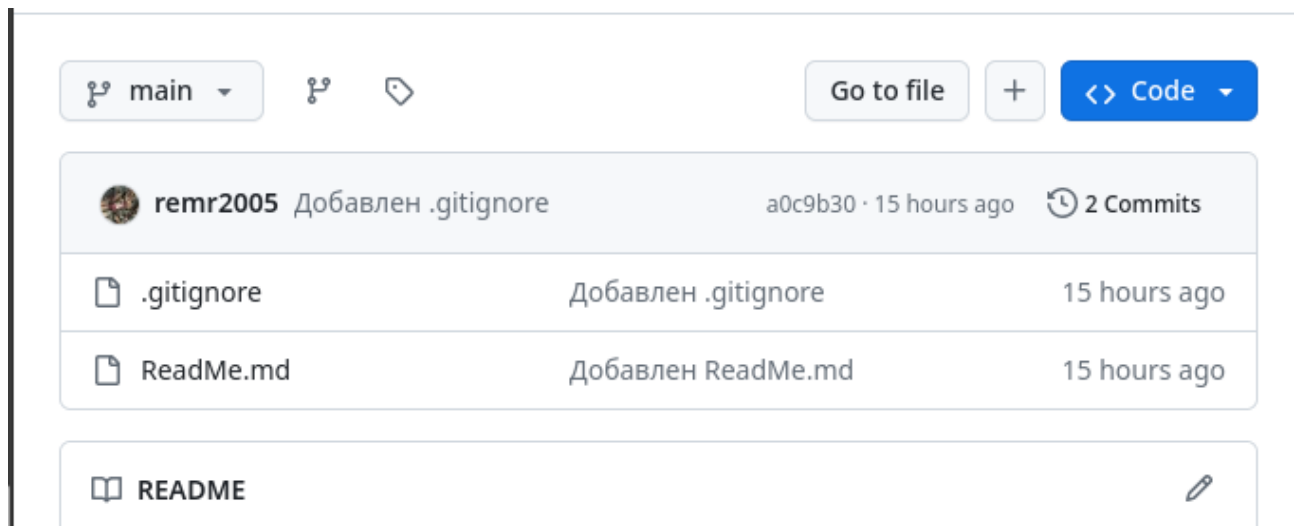
27. нашем случае, локальный репозиторий уже существует, поэтому нужно выполнить всего 4 шага:

В терминале ввести команду: `git remote add origin <ссылка на репозиторий>`. После этой команды гит запомнит ссылку на удалённый репозиторий под именем "origin". Этот шаг нас ни к чему не обязывает, таких ссылок можно запомнить любое количество и куда угодно (хоть на google).

- Затем, GitHub рекомендует переименовать нашу ветку "master" (в которой мы находимся) в "main". Сделаем это: `git branch -M main`.
На самом деле это действие не обязательное, GitHub-у всё равно как называются ваши ветки.
- После чего нужно синхронизировать наш локальный репозиторий и удалённый. Для этого нужно отправить всё коммиты, которые есть у нас на GitHub: `git push -u origin main`. Ключ `-u` назначит удалённый репозиторий "origin", как репозиторий по умолчанию для ветки "main". Т.е. в будущем достаточно просто вводить `git push` без указания куда нужно отправлять коммиты.
Перед отправкой гит запросит учётные данных от удалённого репозитория. Раньше это были логин и пароль от GitHub, но с недавних пор вместо пароля нужно вводить токен (как получить: [\[видео\]](#) [\[текст\]](#)). Во время создания токена выбирайте вариант (classic) и поставьте галочки в разделах `repo` и `workflow`. Токен нам ещё понадобится, поэтому сохраните его где-нибудь, т.к. GitHub вам его больше не покажет.
- В Windows, гит запоминает логин и токен во время первой отправки и больше не спрашивает. Но как правило логин и токен нужно вводить каждый раз. Чтобы упростить себе выполнение работы воспользуйтесь [хранилищем учётных данных](#). Для этого введите команду: `git config --global credential.helper store`.
После этого гит запомнит учётные данные, после первого их введения. Проблем здесь заключается в том, что учётные данных хранятся в файле (по умолчанию `"~/.git-credentials"`) в открытом виде. В принципе можно удалять файл после работы или хранить его на внешнем носителе.

```
kemran@kemran:~/var_keeper$ git remote add origin git@github.com:remr2005/var_keeper.git
kemran@kemran:~/var_keeper$ git branch -M main
kemran@kemran:~/var_keeper$ git push -u origin main
Перечисление объектов: 6, готово.
Подсчет объектов: 100% (6/6), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (5/5), готово.
Запись объектов: 100% (6/6), 2.32 КиБ | 2.32 МиБ/с, готово.
Всего 6 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To github.com:remr2005/var_keeper.git
 * [new branch]      main -> main
Ветка «main» отслеживает внешнюю ветку «main» из «origin».
kemran@kemran:~/var_keeper$ git config --global credential.helper store
```

28. Откройте репозиторий на GitHub. Вы должны увидеть, что "ReadMe.md" и ".gitignore" уже загружены.



29. Введите команду:

```
git branch
```

Гит покажет список веток, которые есть в репозитории. На текущий момент, должна быть только одна ветка - "main".

```
kemran@kemran:~/var_keeper$ git branch
* main
kemran@kemran:~/var_keeper$
```

30. Введите команду:

```
git branch dev
```

- В результате будет создана новая ветка с именем "dev", но при этом вы останетесь в "main".

Внимание: далее по тексту в начале некоторых пунктов будет указано название ветки в которой вы должны находиться *на начало* этого пункта. Это сделано, чтобы можно было проверить, в той ли ветке вы находитесь.

31. [main] Проверьте, что ветка создана при помощи команды:

```
git branch
```

Вы должны увидеть две ветки: "main" и "dev", при этом символом * показана активная ветка.

32. [main] Перейдите в ветку "dev" командой:

```
git checkout dev
```

Убедитесь, что вы действительно в ней.

Команда `git checkout` позволяет переходить не только с ветки на ветку, но и на любой коммит. В этом случае нужно указать его хэш (можно частично), тег или другую метку указывающую на нужный коммит.

```
kemran@kemran:~/var_keeper$ git branch dev
kemran@kemran:~/var_keeper$ git branch
dev
* main
kemran@kemran:~/var_keeper$ git checkout dev
Переключились на ветку «dev»
kemran@kemran:~/var_keeper$
```

33. На данный момент файлы во всех ветках у нас одинаковые. Дальнейшую разработку мы будем вести в ветке "dev", а в ветку "main" будем переносить только полностью готовые и протестированные версии приложения.
34. Обычно, параллельно с разработкой разворачивают инфраструктуру (ПО и сервера) выполняющую специализированные задачи, например: тестирование, предрелизную проверку, демонстрацию работы приложения заказчику и т.д. В нашем случае все эти задачи будут решаться посредством GitHub Actions.
- GitHub Actions работает по принципу: событие -> реакция. Полный список событий можно найти в документации. В качестве реакции на событие запускается рабочий процесс (workflow) выполняющий заданную пользователем последовательность действий на виртуальных машинах.
35. Чтобы познакомиться с основными принципами работы GitHub Actions создадим дополнительную временную ветку в которой опишем несколько простых рабочих процессов (workflow).
36. [dev] Создайте ветку "test" и перейдите в неё. Теперь у нас 3 ветки, и в каждой из них все файлы одинаковые.

```
kemran@kemran:~/var_keeper$ git branch test
kemran@kemran:~/var_keeper$ git checkout test
Переключились на ветку «test»
kemran@kemran:~/var_keeper$
```

37. [test] В корне репозитория создайте каталог ".github". В нём создайте каталог "workflows". В нём будут располагаться файлы с рабочими процессами.

on: push

jobs:

welcome:

runs-on: ubuntu-latest

steps:

- run: echo "Hello new commits"

Название файла может быть любым. В данном файле в разделе **on** описываются все события для которых нужно выполнить этот рабочий процесс. В разделе **jobs** описывается одно или несколько заданий (здесь одно). **welcome** - это произвольное имя задания. **runs-on** - список машин на которых нужно выполнить этот job (в нашем случае только ubuntu-latest). Раздел **steps** - это список действий, которые нужно выполнить в пределах текущего job-a. Здесь мы просим выполнить консольную команду **echo**, но с тем же успехом можно использовать и любые другие консольные команды.

Данный рабочий процесс будет выполняться всегда после того, как будет выполнен **push** в репозиторий на github.

```
kemran@kemran:~/var_keeper$ mkdir .github
kemran@kemran:~/var_keeper$ mkdir .github/workflows
kemran@kemran:~/var_keeper$ nano .github/workflows/welcome.yml
```

38. [test] Перейдите в корень репозитория, затем выполните команду: `ls -al` и убедитесь, что каталог `".github"` присутствует в репозитории.

39. [test] Выполните:

```
git add .
git commit -m "Добавлен workflow welcome"
```

Команда `git add` с точкой в конце добавила в stage зону все файлы и папки, которые находились в *текущем* каталоге и как следствие они попали в наш последний коммит.

```
kemran@kemran:~/var_keeper$ ls -al
итого 28
drwxrwxr-x  4 kemran kemran 4096 мая 11 14:01 .
drwxr-x---+ 60 kemran kemran 4096 мая 11 13:55 ..
drwxrwxr-x  8 kemran kemran 4096 мая 11 13:57 .git
drwxrwxr-x  3 kemran kemran 4096 мая 11 14:02 .github
-rw-rw-r--  1 kemran kemran 5813 мая 10 22:21 .gitignore
-rw-rw-r--  1 kemran kemran  190 мая 10 22:16 ReadMe.md
kemran@kemran:~/var_keeper$ git add .
kemran@kemran:~/var_keeper$ git commit -m "Добавлен workflow welcome"
[test ca91f8c] Добавлен workflow welcome
1 file changed, 6 insertions(+)
create mode 100644 .github/workflows/welcome.yml
kemran@kemran:~/var_keeper$
```

40. [test -> dev] Перейдите в ветку `"dev"` и снова проверьте содержимое репозитория. Каталога `".github"` быть не должно.

41. [dev -> test] Вернитесь обратно в `"test"` и убедитесь, что `".github"` на месте.

```
kemran@kemran:~/var_keeper$ git checkout dev
Переключились на ветку «dev»
kemran@kemran:~/var_keeper$ ls -al
итого 24
drwxrwxr-x  3 kemran kemran 4096 мая 11 14:05 .
drwxr-x---+ 60 kemran kemran 4096 мая 11 13:55 ..
drwxrwxr-x  8 kemran kemran 4096 мая 11 14:05 .git
-rw-rw-r--  1 kemran kemran 5813 мая 10 22:21 .gitignore
-rw-rw-r--  1 kemran kemran  190 мая 10 22:16 ReadMe.md
kemran@kemran:~/var_keeper$ git checkout test
Переключились на ветку «test»
kemran@kemran:~/var_keeper$ ls -al
итого 28
drwxrwxr-x  4 kemran kemran 4096 мая 11 14:05 .
drwxr-x---+ 60 kemran kemran 4096 мая 11 13:55 ..
drwxrwxr-x  8 kemran kemran 4096 мая 11 14:05 .git
drwxrwxr-x  3 kemran kemran 4096 мая 11 14:05 .github
-rw-rw-r--  1 kemran kemran 5813 мая 10 22:21 .gitignore
-rw-rw-r--  1 kemran kemran  190 мая 10 22:16 ReadMe.md
kemran@kemran:~/var_keeper$
```

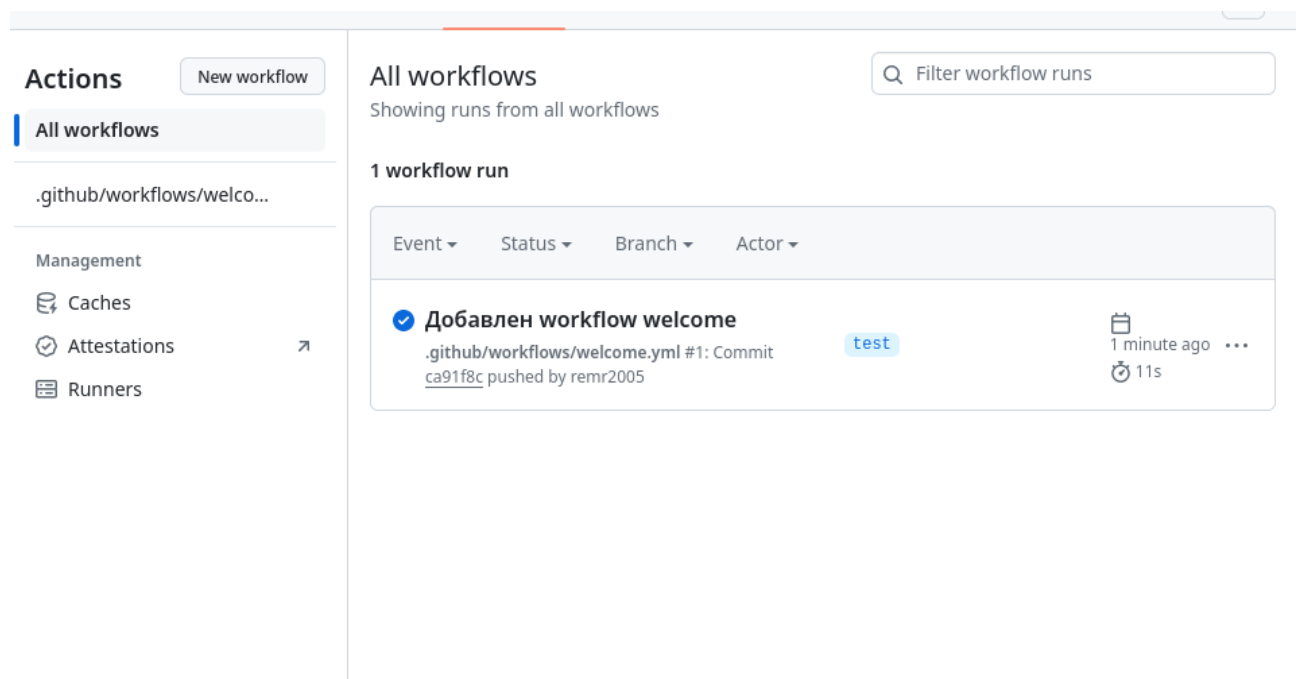
42. [test] Чтобы созданный нами рабочий процесс выполнялся его нужно отправить на GitHub. Но, на данный момент, удалённый репозиторий связан только с веткой `"main"`,

и git не знает куда ему нужно отправить ветку "test". Снова воспользуемся ключом -u и укажем origin как место назначения:

```
git push -u origin test
```

```
kemran@kemran:~/var_keeper$ git push -u origin test
Перечисление объектов: 6, готово.
Подсчет объектов: 100% (6/6), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (5/5), 512 байтов | 512.00 КиБ/с, готово.
Всего 5 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:   https://github.com/remr2005/var_keeper/pull/new/test
remote:
To github.com:remr2005/var_keeper.git
 * [new branch]      test -> test
Ветка «test» отслеживает внешнюю ветку «test» из «origin».
kemran@kemran:~/var_keeper$
```

43. Перейдите на GitHub в репозиторий "var_keeper" и откройте раздел "Actions". Слева вы увидите список существующих в репозитории рабочих процессов (в нашем случае только .github/workflows/welcome.yml). Основную часть экрана занимает список запущенных/выполненных процессов.



44. К этому моменту наш процесс должен был уже завершиться и рядом с ним должна быть зелёная галочка. Щёлкните по нему и откроется окно визуализирующее взаимосвязи job-ов в процессе. У нас он только один.

✓ **Добавлен workflow welcome #1**

Re-run all jobs ⋮

🏠 **Summary**

Jobs

✓ welcome

Run details

🕒 Usage

📄 Workflow file

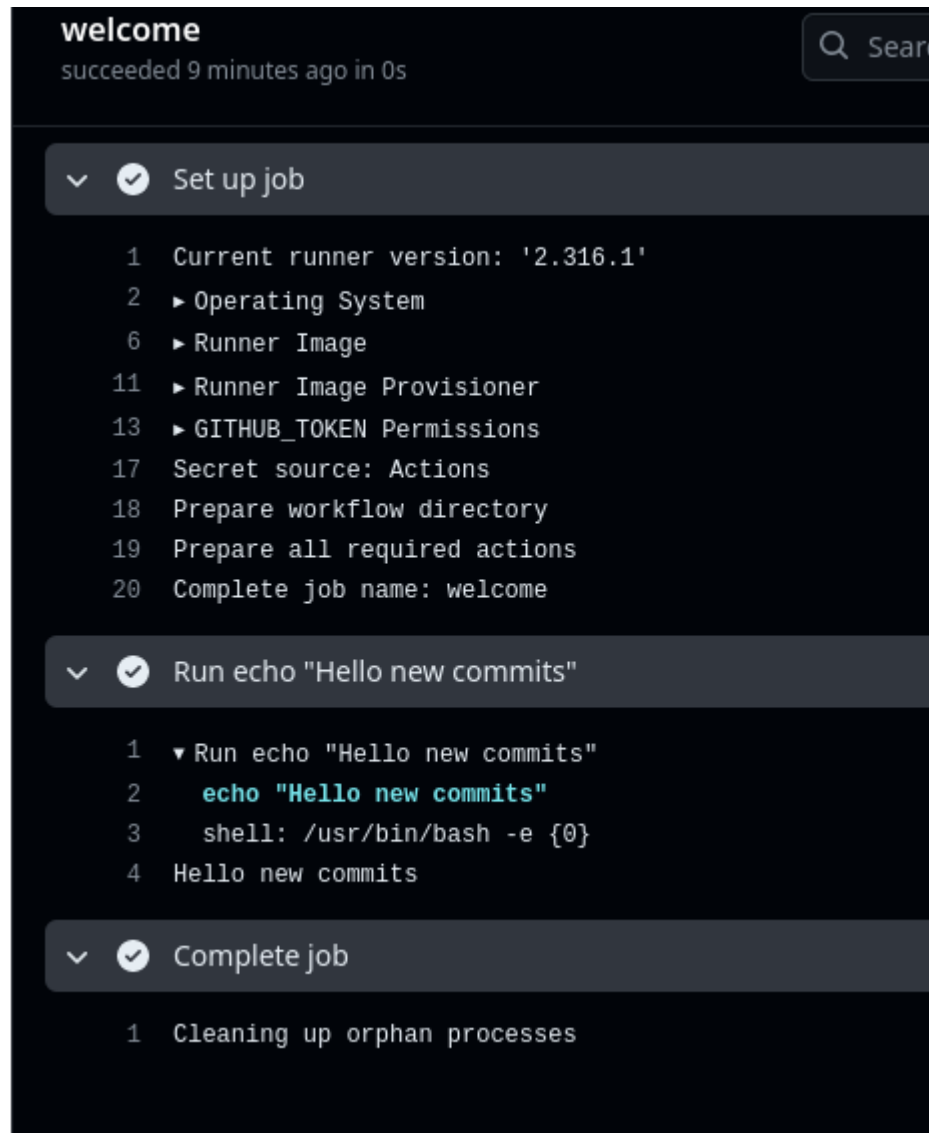
Triggered via push 2 minutes ago	Status	Total duration	Artifacts
👤 remr2005 pushed → ca91f8c test	Success	11s	-

welcome.yml
on: push

✓ welcome0s

📄 - +

45. Щёлкните по нему и откроется более подробная информация о выполнении процесса. Развернув раздел под названием Run echo "Hello new commits" вы увидите результат работы команды, т.е. само сообщение "Hello new commits".



46. [test] Обратите внимание, что рабочий процесс в списке назван по полному имени файла, а в списке запущенных процессов он значится под именем коммита в котором был добавлен. Исправим это.

Откройте, в локальном репозитории, файл "welcome.yml" и добавьте в начало:

```
name: Welcome workflow
run-name: Welcome new commits
```

Параметр `name` определяет имя в списке рабочих процессов, а `run-name` в списке запущенных.

47. [test] Закоммитьте и отправьте изменения на GitHub:

```
git add .
git commit -m "Обновлён workflow welcome"
git push
```

Посмотрите, что изменилось в разделе "Actions".

```
kemran@kemran:~/var_keeper$ nano .github/workflows/welcome.yml
kemran@kemran:~/var_keeper$ git add .
git commit -m "Обновлён workflow welcome"
git push
[test e79a620] Обновлён workflow welcome
1 file changed, 2 insertions(+)
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (5/5), 533 байта | 533.00 КиБ/с, готово.
Всего 5 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To github.com:remr2005/var_keeper.git
ca91f8c..e79a620 test -> test
kemran@kemran:~/var_keeper$
```

2 workflow runs

Event ▾	Status ▾	Branch ▾	Actor ▾
✓ Welcome new commits			
Welcome workflow #2: Commit e79a620 pushed by remr2005		test	<div>📅 1 minute ago ...</div> <div>🕒 13s</div>
✓ Добавлен workflow welcome			
Welcome workflow #1: Commit ca91f8c pushed by remr2005		test	<div>📅 12 minutes ago ...</div> <div>🕒 11s</div>

48. В каждый workflow GitHub передаёт дополнительную информацию в виде переменных окружения виртуальной машины и в виде [контекстов](#). Список переменных окружения можно посмотреть обычным образом (консольная команда `env`). Для просмотра содержимого контекстов можно попросить одно конкретное значение или воспользоваться функцией `toJson`.

```
kemran@kemran: ~/var_keeper$ env
SHELL=/bin/bash
SESSION_MANAGER=local/kemran:@/tmp/.ICE-unix/2396,unix/kemran:/tmp/.ICE-unix/2396
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-pantheon:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK_IM_MODULE=ibus
LANGUAGE=ru_UA:ru
MANDATORY_PATH=/usr/share/gconf/pantheon.mandatory.path
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=pantheon
GTK_MODULES=gail:atk-bridge
PWD=/home/kemran/var_keeper
LOGNAME=kemran
XDG_SESSION_DESKTOP=pantheon
QT_QPA_PLATFORMTHEME=gtk3
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
SYSTEMD_EXEC_PID=2562
XAUTHORITY=/run/user/1000/gdm/Xauthority
QT_STYLE_OVERRIDE=adwaita
WINDOWPATH=2
HOME=/home/kemran
USERNAME=kemran
IM_CONFIG_PHASE=1
LANG=ru_UA.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:s
g=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha
=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz
=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=
01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.al
z=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.e
sd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;
35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx
=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:
*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;3
5:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;3
5:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=0
0;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=Pantheon
VTE_VERSION=6800
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/7ab480e7_317e_46c8_b7ce_1bb891682275
CLUTTER_IM_MODULE=ibus
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
DEFAULTS_PATH=/usr/share/gconf/pantheon.default.path
LESSOPEN=| /usr/bin/lesspipe %s
LIBVIRT_DEFAULT_URI=qemu:///system
USER=kemran
GNOME_TERMINAL_SERVICE=:1.138
DISPLAY=:0
SHLVL=1
```

49. [test] В каталоге ".github/workflows/" создайте файл "dump_contexts.yml" со следующим содержимым:

```
name: Context testing
on: push
jobs:
  dump_contexts:
    runs-on: ubuntu-latest
    steps:
      - name: Dump GitHub context
        id: github_context_step
        run: echo '${{ toJSON(github) }}'
      - name: Dump job context
        run: echo '${{ toJSON(job) }}'
      - name: Dump steps context
        run: echo '${{ toJSON(steps) }}'
      - name: Dump runner context
        run: echo '${{ toJSON(runner) }}'
```


- name: Dump Secrets context
run: echo '\${{ toJSON(secrets) }}'
- name: Dump strategy context
run: echo '\${{ toJSON(strategy) }}'
- name: Dump matrix context
run: echo '\${{ toJSON(matrix) }}'
- name: Dump environments variable
run: env

Этот workflow, тоже будет выполнен по событию push, но теперь у нас job состоит из 8-ми шагов. Каждый из них выведет информацию о своём контексте на экран, а последний покажет переменные среды.

Чтобы получить конкретное значение из контекста оно должно быть указано в специальном виде: `${{ runner.os }}`. Например эта запись будет заменена на значение ключа "os" контекста "runner".

```
kemran@kemran:~/var_keeper$ nano .github/workflows/dump_contexts.yml
kemran@kemran:~/var_keeper$
```

50. [test] Закоммитьте и отправьте изменения на GitHub. Затем откройте раздел "Actions" и изучите вывод последнего workflow. Обратите внимание, что "Welcome workflow" тоже был запущен.

```
kemran@kemran:~/var_keeper$ git add .
kemran@kemran:~/var_keeper$ git commit -m "Laba is so boring"
[test 439ca1d] Laba is so boring
1 file changed, 23 insertions(+)
create mode 100644 .github/workflows/dump_contexts.yml
kemran@kemran:~/var_keeper$ git push -u origin test
Перечисление объектов: 8, готово.
Подсчет объектов: 100% (8/8), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (4/4), готово.
Запись объектов: 100% (5/5), 665 байтов | 665.00 КиБ/с, готово.
Всего 5 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
To github.com:remr2005/var_keeper.git
e79a620..439ca1d test -> test
Ветка «test» отслеживает внешнюю ветку «test» из «origin».
kemran@kemran:~/var_keeper$
```

Actions

New workflow

All workflows

Context testing

Welcome workflow

Management

Caches

Attestations

Runners

All workflows

Showing runs from all workflows

4 workflow runs

Event	Status	Branch	Actor
✓ Welcome new commits			
Welcome workflow #3: Commit <code>439ca1d</code> pushed by remr2005			
	test		2 minutes ago ... 13s
✓ Laba is so boring			
Context testing #1: Commit <code>439ca1d</code> pushed by remr2005			
	test		2 minutes ago ... 17s
✓ Welcome new commits			
Welcome workflow #2: Commit <code>e79a620</code> pushed by remr2005			
	test		13 minutes ago ... 13s
✓ Добавлен workflow welcome			
Welcome workflow #1: Commit <code>ca91f8c</code> pushed by remr2005			
	test		25 minutes ago ... 11s

51. [test] Ветка "test" нам больше не нужна, поэтому можем её удалить.

Здесь может показаться, что удаление ветки приведёт к удалению коммитов созданных в ней. На самом деле это не так. Понятие *ветка* в git это просто *ссылка* на последний коммит созданный в ней и как только вы создаёте новый коммит в ветке, эта ссылка автоматически перепрыгивает на него. Удаление ветки сводится к удалению этой самой ссылки и не затрагивает коммиты.

52. [test -> dev] Удаление веток в локальном и удалённом репозитории не связано, поэтому их придётся удалять отдельно. Удалить активную ветку нельзя, поэтому сначала перейдите в ветку "dev", а затем выполните команды:

```
git branch -D test
git push origin -d test
```

Обычно для удаления локальной ветки используется `-d`, но нам пришлось использовать усиленную версию `-D` по следующей причине: те три коммита, которые мы делали в ветке "test" всё ещё существуют, но теперь они не доступны и вскоре будут окончательно удалены

гитом. Обычно перед удалением ветки её вливают в другую, в этом случае, даже после удаления ветки, коммиты остаются доступны через коммит слияния.

```
kemran@kemran:~/var_keeper$ git branch -D test
git push origin -d test
error: Нельзя удалить ветку «test» т.к. она активна на «/home/kemran/var_keeper»
To github.com:remr2005/var_keeper.git
- [deleted]      test
kemran@kemran:~/var_keeper$ git checkout dev
Переключились на ветку «dev»
kemran@kemran:~/var_keeper$ git branch -D test
Ветка test удалена (была 439ca1d).
kemran@kemran:~/var_keeper$
```

53. Почему три коммита из удалённой ветки "test" вдруг стали недоступны?

Дело в том, что каждый коммит помнит только своих предков, но не потомков, поэтому находясь на каком-нибудь коммите, можно пройти назад, по всем предыдущим, до самого начала, но не вперёд. После удаления ветки "test", у пользователя больше нет возможности перейти на коммиты созданные в ней, через другую ветку или как-нибудь ещё и поэтому гит считает их мусорными.

На самом деле есть способ попасть на коммиты из удалённой ветки если вы знаете их хэши. Именно поэтому git, при удалении ветки, печатает хэш её последнего коммита и не уничтожает недоступные коммиты сразу же. Зная хэш коммита вы можете перейти на него и восстановить ветку, если удалили её не намеренно.

54. [dev] Создайте новую ветку с названием "template" и перейдите в неё. В этой ветке мы создадим нулевую версию нашего приложения.

```
kemran@kemran:~/var_keeper$ git branch template
kemran@kemran:~/var_keeper$ git checkout template
Переключились на ветку «template»
kemran@kemran:~/var_keeper$
```

55. [template] Настроим локальное окружение, чтобы иметь возможность запускать код у себя на машине. Для этого введите команду: `python3 -m venv venv`. В итоге в корне репозиторий будет создан каталог "venv" с виртуальным окружением. Т.к. это наши рабочие файлы, то они не должны попадать в общий репозиторий. Если вы откроете файл ".gitignore", то в разделе "# Environments" найдёте несколько разных имён, в том числе и `venv/`. Это означает, что дополнительных изменений нам вносить не нужно, т.к. гит уже знает, что эту папку нужно игнорировать.

56. [template] Активируйте виртуальное окружение, т.к. же как вы это делали ранее: `source venv/bin/activate`.

57. [template] Установите необходимые для работы пакеты: `pip install flask`

```
kemran@kemran:~/var_keeper$ python3 -m venv venv
kemran@kemran:~/var_keeper$ source venv/bin/activate
(venv) kemran@kemran:~/var_keeper$ pip install flask
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
    101.7/101.7 KB 787.3 kB/s eta 0:00:00
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.3-py3-none-any.whl (227 kB)
    227.3/227.3 KB 1.9 MB/s eta 0:00:00
Collecting blinker>=1.6.2
  Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
    97.9/97.9 KB 1.7 MB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting Jinja2>=3.1.2
  Downloading jinja2-3.1.4-py3-none-any.whl (133 kB)
    133.3/133.3 KB 1.7 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
Installing collected packages: MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, flask
Successfully installed Jinja2-3.1.4 MarkupSafe-2.1.5 Werkzeug-3.0.3 blinker-1.8.2 click-8.1.7 flask-3.0.3 itsdangerous-2.2.0
(venv) kemran@kemran:~/var_keeper$
```

58. [template] Создайте в корне репозитория каталог "src", внутри которого создайте каталог "app". В каталоге "app" создайте файл "app.py" с содержимым:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0')
```

Это шаблон будущего проекта (нулевая версия).

```
(venv) kemran@kemran:~/var_keeper$ mkdir src
(venv) kemran@kemran:~/var_keeper$ mkdir src/app
(venv) kemran@kemran:~/var_keeper$ nano src/app/app.py
(venv) kemran@kemran:~/var_keeper$
```

59. [template] Запустите приложение и, в новом терминале, проверьте его работоспособность, при помощи команды: `curl http://127.0.0.1:5000/`. `curl` отправит GET-запрос на `http://127.0.0.1:5000/`. В результате вы должны увидеть сообщение `Hello, World!`.

Если всё хорошо, то остановите приложение и закройте новый терминал.

```
kemran@kemran:~/var_keeper$ curl http://127.0.0.1:5000/
Hello, World!kemran@kemran:~/var_keeper$
```

60. [template] Сохраните зависимости приложения в файл в **корень** репозитория:

```
pip freeze > requirements.txt
```

```
(venv) kemran@kemran:~/var_keeper$ pip freeze > requirements.txt
(venv) kemran@kemran:~/var_keeper$
```

61. [template] Находясь в корне репозитория наберите команду: `git status`. Гит должен показать, что присутствуют неотслеживаемые файлы. Среди этих файлов должен быть только каталог "src" и "requirements.txt". Каталога "venv" в списке быть не должно, т.к. он игнорируется гитом.

62. [template] Выполните:

```
git add .
git commit -m "Добавлен шаблон приложения"
```

```
(venv) kemran@kemran:~/var_keeper$ git status
Текущая ветка: template
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
  requirements.txt
  src/

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
(venv) kemran@kemran:~/var_keeper$ git add .
git commit -m "Добавлен шаблон приложения"
[template 8adff81] Добавлен шаблон приложения
 2 files changed, 17 insertions(+)
 create mode 100644 requirements.txt
 create mode 100644 src/app/app.py
(venv) kemran@kemran:~/var_keeper$
```

63. Обычно, параллельно с разработкой приложения разрабатываются и тесты, которые проверяют, что при добавлении нового функционала ничего не сломалось (регрессионное тестирование). Т.к. разработчик постоянно меняет код, то и тесты тоже должны запускаться постоянно. Следовательно они должны выполняться максимально быстро, чтобы не тормозить процесс. Быстро проверить всё приложение не получится, поэтому ограничиваются проверкой отдельных модулей (юнит тесты). В этой работе не будет модульных тестов, но мы будем делать вид, что они есть.

64. [template] В корне репозитория создайте папку "test" и в нём создайте файл "unit-tests.sh" со следующим содержимым:

```
echo "Unit Tests PASS"
Это обычный shell скрипт.
```

65. [template] Выполните:

```
git add .
```

```
git commit -m "Добавлены unit тесты"
```

```
(venv) kemran@kemran:~/var_keeper$ mkdir test
(venv) kemran@kemran:~/var_keeper$ nano test/unit-tests.sh
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Добавлены unit тесты"
[template cc9d5d5] Добавлены unit тесты
1 file changed, 1 insertion(+)
create mode 100644 test/unit-tests.sh
(venv) kemran@kemran:~/var_keeper$
```

66. Добавим workflow, который будет запускать эти тесты при каждом push. Если в качестве триггера указать просто push, то тесты будут запускаться при push в любую ветку в которую мы вольём нашу текущую, т.к. workflow попадёт и туда тоже. По итогу работы мы должны будем влить эту ветку в "dev", а "dev" рано или поздно вольётся в "main". По причинам, которые будут понятны далее мы не хотим запускать тесты при push в "dev" и "main". Они должны запускаться только при push в ветки, в которых разрабатываются отдельные фичи (как наша). Поэтому в workflow исключим "dev" и "main".

67. [template] Т.к. каталог ".github/workflows/" мы создавали только в ветке "test", а затем её удалили, то на данный момент этих каталогов у нас нет. Создайте их.

68. [template] В ".github/workflows/" создайте файл: "unit_tests.yml" содержащий:

```
name: Unit Tests
run-name: Run Unit Tests
on:
  push:
    branches-ignore:
      - 'main'
      - 'dev'
jobs:
  unit_testing:
    runs-on: ubuntu-latest
    steps:
      - name: Run Unit Tests
        run: |
          chmod +x ./test/unit-tests.sh
          ./test/unit-tests.sh
        shell: bash
```

Вертикальная черта после run указывает, что далее будет многострочный текстовый литерал, без неё, перевод строки был бы воспринят как конец значения и начало нового ключа. Параметр shell указывает какой программе следует отдать текст из раздела run. В

данном случае это терминал bash, но в принципе можно указать например интерпретатор python, в этом случае run должен содержать python скрипт.

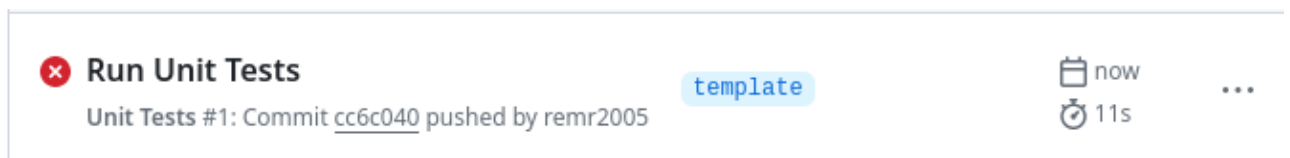
Здесь `chmod +x ./test/unit-tests.sh` добавляет права на исполнение файлу `./test/unit-tests.sh` (путь от корня репозитория). Следующая строка запускает файл как программу.

```
(venv) kemran@kemran:~/var_keeper$ mkdir .github/
(venv) kemran@kemran:~/var_keeper$ mkdir .github/workflows
(venv) kemran@kemran:~/var_keeper$ nano .github/workflows/unit_tests.yml
(venv) kemran@kemran:~/var_keeper$
```

69. [template] Закоммитьте изменения и отправьте на GitHub. Ветка "template" новая, поэтому гиту нужно будет указать куда её отправлять.

```
(venv) kemran@kemran:~/var_keeper$ git push -u origin template
Перечисление объектов: 11, готово.
Подсчет объектов: 100% (11/11), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (10/10), 1.00 КиБ | 1.00 МиБ/с, готово.
Всего 10 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), done.
remote:
remote: Create a pull request for 'template' on GitHub by visiting:
remote:   https://github.com/remr2005/var_keeper/pull/new/template
remote:
To github.com:remr2005/var_keeper.git
 * [new branch]      template -> template
Ветка «template» отслеживает внешнюю ветку «template» из «origin».
(venv) kemran@kemran:~/var_keeper$
```

70. Перейдите на GitHub в раздел "Actions". Вы должны увидеть, что процесс завершился с ошибкой. Если вы посмотрите более подробную информацию по процессу, то вы поймёте, что проблема появились на этапе "Run Unit Tests" и в тексте ошибки сказано: "No such file or directory". Дело в том, что по умолчанию в рабочем каталоге виртуальной машины запущенной для выполнения workflow, ничего нет, в том числе и наших файлов из репозитория.



71. Чтобы скопировать файлы из репозитория в рабочий каталог виртуальной машины можно воспользоваться обычным способом (клонировать репозиторий командой `git clone`). Но GitHub предоставляет более удобный способ (заранее подготовленные действия Actions). В нашем случае перед шагом "Run Unit


Tests" нужно добавить дополнительный шаг (не забывайте про отступы):

- name: Checkout
uses: actions/checkout@v3

Здесь uses говорит, что этот шаг должен выполнить Action расположенный в репозитории actions/checkout (<https://github.com/actions/checkout>) и при этом использовать версию с тегом v3. Это действие без параметров, как раз и копирует файлы из репозитория в рабочий каталог виртуальной машины. При желании можно указать какой коммит или ветку требуется копировать.



```
kemran@kemran:~/var_keeper$ nano .github/workflows/unit_tests.yml
kemran@kemran:~/var_keeper$ git add .
kemran@kemran:~/var_keeper$ git commit -m "Fix mistake in unit-tests"
[template 2a1d6fb] Fix mistake in unit-tests
1 file changed, 2 insertions(+)
kemran@kemran:~/var_keeper$ git push origin template
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (5/5), 438 байтов | 438.00 КиБ/с, готово.
Всего 5 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:remr2005/var_keeper.git
cc6c040..2a1d6fb template -> template
kemran@kemran:~/var_keeper$
```

72. [template] Добавьте указанный выше шаг в "unit_tests.yml", сделайте новый коммит и push. Теперь если вы перейдете в раздел "Actions", то обнаружите, что процесс завершился успешно и на шаге "Run Unit Tests" было выведено сообщение: "Unit Tests PASS".

 **Run Unit Tests**

Unit Tests #2: Commit 2a1d6fb pushed by remr2005

template

 1 minute ago ...
 11s

73. Теперь в процессе разработки по push в любую ветку кроме "dev" и "main" будут запускаться наши воображаемые unit тесты.

74. Будем считать, что мы закончили добавление новой фичи в проект и привели его в рабочее состояние. Следующим шагом должно быть отправка нашего кода в ветку "dev" (слияние веток). Но, как правило, только unit тестов не достаточно чтобы убедиться в полной работоспособности проекта, т.к. по отдельности модули могут работать, а в готовом приложении нет. Добавим интеграционный тест, который будет проверять

работоспособность приложения в целом.

На каком же этапе нужно выполнить этот тест? Очевидно, что после добавления кода из "template" в "dev" это делать уже поздно, т.к. сломанный код уже попал в ветку, следовательно нужно это сделать раньше. С другой стороны разворачивать окружение для интеграционного тестирования локально может быть очень сложно и не удобно, поэтому хотелось бы это сделать удалённо.

75. На GitHub (не в git) есть механизм который называется pull request. Если по простому, то он состоит из 2х шагов:

- push того, что получилось после слияния в "dev". Этот шаг не выполнится, пока пользователь не разрешит.
- pull из ветки в которую хотим влиться в ветку которую хотим влить. То есть как бы обратный шаг. Мы берём все изменения из "dev" и вливаем их в "template".

Результат будет именно тот, что мы хотели изначально, т.е. наш код из "template" попадёт в "dev", но благодаря первому шагу мы *заранее* получим слитое состояние веток "template" и "dev", которое и сможем протестировать перед тем как разрешить второй шаг.

76. [template] В каталоге ".github/workflows/" создайте файл "pull-request-to-dev.yml" содержащий:

```
name: Pull Request To Dev
run-name: Run Integration Tests
on:
  pull_request:
    branches:
      - 'dev'
jobs:
  integration_testing:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Install requirements
        run: pip install -r requirements.txt
      - name: Run app
        run: python3 ./src/app/app.py &
      - name: Test GET-request
        run: |
          ANSW=$(curl http://127.0.0.1:5000/)
          if [ "$ANSW" != "Hello, World!" ]; then
            exit 1
          else
            echo "Integration Test PASS"
          fi
```

В этом рабочем процессе мы устанавливаем все необходимые для работы приложения "app.py" зависимости из файла "requirements.txt". На шаге "Run app" запускаем приложение и на шаге "Test GET-request" выполняем shell скрипт, который обращается к нашему

приложению `curl http://127.0.0.1:5000/` и сохраняет результат работы в переменную `ANSW`, затем значение `ANSW` сравнивается с текстом "Hello, World!" и в случае неравенства скрипт завершается с ненулевым кодом (т.е. ошибка). Если значения совпадают, пишем "Integration Test PASS".

В качестве триггера для процесса выбираем `pull_request`. Здесь ограничим только `pull_request` в "dev", т.к. для ветки `main` будут свой workflow.

77. [template] Закоммитьте и отправьте на GitHub изменения.

```
(venv) kemran@kemran:~/var_keeper$ nano .github/workflows/pull-request-to-dev.yml
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Pull request to dev"
[template a6c6cf8] Pull request to dev
1 file changed, 24 insertions(+)
create mode 100644 .github/workflows/pull-request-to-dev.yml
(venv) kemran@kemran:~/var_keeper$ git push origin template
Перечисление объектов: 8, готово.
Подсчет объектов: 100% (8/8), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (4/4), готово.
Запись объектов: 100% (5/5), 734 байта | 734.00 КиБ/с, готово.
Всего 5 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:remr2005/var_keeper.git
2a1d6fb..a6c6cf8 template -> template
(venv) kemran@kemran:~/var_keeper$
```

78. [template] Прежде, чем создать pull request мы должны отправить ветку "dev" на GitHub, т.к. она пока существует только в локальной репозитории. Как обычно выполните команду `git push -u origin dev` чтобы гит отправил и запомнил куда отправлять коммиты в будущем. Переходить в ветку "dev" не обязательно.


```
(venv) kemran@kemran:~/var_keeper$ git push -u origin dev
Всего 0 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:   https://github.com/remr2005/var_keeper/pull/new/dev
remote:
To github.com:remr2005/var_keeper.git
* [new branch]      dev -> dev
Ветка «dev» отслеживает внешнюю ветку «dev» из «origin».
(venv) kemran@kemran:~/var_keeper$
```

79. Откройте репозиторий на GitHub и в разделе "Pull requests" нажмите кнопку "New pull request":

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#). [Learn more about diff comparisons here](#).

base: dev ← compare: template ✓ **Able to merge.** These branches can be automatically merged.



Add a title

Add a description

Write Preview H B I ≡ < > 🔗 ☰ ☷ 🔍 @ ...

Add your description here...

Markdown is supported Paste, drop, or click to add files

Create pull request ▾

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Reviewers

No reviews

Assignees

No one—[assign yourself](#)

Labels

None yet

Projects

None yet

Milestone

No milestone

Development

Use [Closing keywords](#) in the description to automatically close issues

Helpful resources

[GitHub Community Guidelines](#)

5 commits

5 files changed

1 contributor

Commits on May 11, 2024

80. Выберите для слияния ветки "dev" и "template" и нажмите "Create pull request":

81. Укажите название pull request-а и нажмите "Create pull request":

82. Теперь pull request создан и под списком коммитов можно увидеть выполненные workflow. Если хоть один из них не закончится успешно pull request нельзя будет завершить. Нажмите на Details возле workflow "Pull Request To Dev" и проверьте, что на шаге "Test GET-request" было выведено сообщение "Integration Test PASS":

← Back to pull request #1

✓ Run Integration Tests #1

Re-run all jobs

...

Summary

Jobs

✓ integration_testing

Run details

Usage

Workflow file

integration_testing

succeeded 1 minute ago in 10s

Search logs



- > ✓ Set up job
- > ✓ Checkout
- > ✓ Install requirements
- > ✓ Run app






✓ Test GET-request

1 ▶ Run ANSW=\$(curl http://127.0.0.1:5000/)

```
9 % Total % Received % Xferd Average Speed Time Time Time Current
10 % Dload Upload Total Spent Left Speed
11
12 0 0 0 0 0 0 0 0 0 0 0 0
13 100 13 100 13 0 0 8660 0 0 0 13000
14 Integration Test PASS
```

- > ✓ Post Checkout
- > ✓ Complete job

83. Чтобы завершить pull request нужно выбрать один из вариантов слияния веток. Обычно стараются выбирать Rebase, чтобы история разработки была линейная. Но мы выберем первый вариант, чтобы посмотреть что получится. Нажмите "Merge pull request", а затем подтвердите действие:

○	 Добавлен шаблон приложения	8adff81
○	 Добавлены unit тесты	cc9d5d5
○	 s	✗ cc6c040
○	 Fix mistake in unit-tests	✓ 2a1d6fb
○	 Pull request to dev	✓ a6c6cf8

Add more commits by pushing to the **template** branch on **remr2005/var_keeper**.



Merge pull request #1 from remr2005/template

Template

This commit will be authored by 109353270+remr2005@users.noreply.github.com

Confirm merge

Cancel



Add a comment

Write

Preview

H

B

I

≡

<>

🔗

≡

≡

📎

@

...

Add your comment here...

Markdown is supported

Paste, drop, or click to add files

Close pull request

Comment

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

84. Ветка "template" выполнила свою роль и больше не нужна. Удалите её. При этом коммиты, которые были в этой ветке не удалятся сборщиком мусора git



remr2005 deleted the **template** branch now

Restore branch

85. [template -> dev] Теперь на GitHub присутствуют коммиты, которых нет у нас в локальном репозитории (коммит слияния). Если мы сейчас продолжим разработку, то в дальнейшем

возникнут проблемы при отправке коммитов на GitHub. Чтобы этого избежать, в терминале, перейдите на ветку "dev" и скачайте изменения в локальный репозиторий: `git pull`.

```
(venv) kemran@kemran:~/var_keeper$ git checkout dev
Переключились на ветку «dev»
Эта ветка соответствует «origin/dev».
(venv) kemran@kemran:~/var_keeper$ git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Распаковка объектов: 100% (1/1), 891 байт | 891.00 КиБ/с, готово.
Из github.com:remr2005/var_keeper
   a0c9b30..3b7cae2  dev      -> origin/dev
Обновление a0c9b30..3b7cae2
Fast-forward
 .github/workflows/pull-request-to-dev.yml | 24 ++++++
 .github/workflows/unit_tests.yml          | 18 ++++++
 requirements.txt                          |  7 ++++++
 src/app/app.py                            | 10 ++++++
 test/unit-tests.sh                        |  1 +
5 files changed, 60 insertions(+)
create mode 100644 .github/workflows/pull-request-to-dev.yml
create mode 100644 .github/workflows/unit_tests.yml
create mode 100644 requirements.txt
create mode 100644 src/app/app.py
create mode 100644 test/unit-tests.sh
(venv) kemran@kemran:~/var_keeper$
```

86. [dev] Удалите локальную ветку "template" и с GitHub тоже:

```
git branch -d template
git push origin -d template
```

Обратите внимание, что теперь мы используем не усиленную -d.

```
(venv) kemran@kemran:~/var_keeper$ git branch -d template
git push origin -d template
Ветка template удалена (была a6c6cf8).
error: unable to delete 'template': remote ref does not exist
error: не удалось отправить некоторые ссылки в «github.com:remr2005/var_keeper.git»
(venv) kemran@kemran:~/var_keeper$
```

87. [dev] Посмотрите историю репозитория при помощи команды: `git log --oneline --graph`. Как видно, в истории присутствует "петля", которая появилась в результате добавления merge-коммита. Чтобы предотвратить появление таких петель и используется стратегия "Rebase and merge":

```
(venv) kemran@kemran:~/var_keeper$ git log --oneline --graph
* 3b7cae2 (HEAD -> dev, origin/dev) Merge pull request #1 from remr2005/template
| \
| * a6c6cf8 (origin/template) Pull request to dev
| * 2a1d6fb Fix mistake in unit-tests
| * cc6c040 s
| * cc9d5d5 Добавлены unit тесты
| * 8adff81 Добавлен шаблон приложения
| /
* a0c9b30 (origin/main, main) Добавлен .gitignore
* 00f763a Добавлен ReadMe.md
(venv) kemran@kemran:~/var_keeper$
```

88. Т.к. в ветку "dev" попадает только полностью рабочая версия приложения, почти готовая к релизу, хотелось бы иметь возможность "пощупать" его в виде собранного приложения (stage версия). Для этого запакуем его в docker контейнер и отправим в специальный репозиторий, из которого, можно будет получить контейнер с приложением обычным образом.

89. [dev -> docker] Новый функционал будем добавлять в отдельной ветке. Для этого создайте и перейдите в ветку "docker".

90. [docker] Обновление stage версии будем выполнять автоматически, каждый раз, когда будет обновятся ветка "dev" (т.е. по push в неё). Для этого в каталоге ".github/workflows/" создайте файл "staging.yml" содержащий:

```
name: Push Stage version to DockerHub
on:
  push:
    branches:
      - 'dev'
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Cut commit sha
        id: cut
        run: echo "sha_short=${GITHUB_SHA::7}" >> $GITHUB_OUTPUT

      # Login against a Docker registry
      # https://github.com/docker/login-action
      - name: Log into DockerHUB
        uses: docker/login-action@28218f9b04b4f3f62068d7b6ce6ca5b26e35336c
        with:
          username: ${ secrets.DOCKER_USERNAME }
          password: ${ secrets.DOCKER_TOKEN }

      - name: Setup Docker buildx
        uses: docker/setup-buildx-
action@79abd3f86f79a9d68a23c75a09a9a85889262adf

      # Build and push Docker image with Buildx
      # https://github.com/docker/build-push-action
```

```

- name: Build and push Docker image
  id: build-and-push
  uses: docker/build-push-action@ac9327eae2b366085ac7f6a2d02df8aa8ead720a
  with:
    context: .
    file: ./docker/Dockerfile
    push: true
    tags: ${{ vars.DOCHUB_USERNAME }}/var_keeper:$
  {{ steps.cut.outputs.sha_short }}

```

```

venv) kemran@kemran:~/var_keeper$ git branch docker
venv) kemran@kemran:~/var_keeper$ git checkout docker
переключились на ветку «docker»
venv) kemran@kemran:~/var_keeper$ nano .github/workflows/staging.yml
venv) kemran@kemran:~/var_keeper$

```

Здесь используется 4 действия `actions/checkout` - чтобы получить содержимое репозитория в рабочий каталог виртуальной машины; `docker/login-action` - чтобы залогиниться на DockerHUB; `docker/setup-buildx-action` - настройка "buildx" - плагина Docker CLI для расширенных возможностей сборки с помощью BuildKit; `docker/build-push-action` - сборка и отправка образа в репозиторий DockerHUB. После символа @ указаны sha коммитов или тег коммита.

Для того, чтобы этот workflow отработал корректно нужно:

- Создать Dockerfile, на основании которого будет собираться образ. В нашем workflow указано, что Dockerfile лежит в каталоге "docker", который расположен в корне репозитория.
- Добавить логин и пароль от аккаунта DockerHub в раздел [секретов GitHub репозитория](#). Имя пользователя должно быть сохранено под именем DOCHUB_USERNAME, пароль под именем DOCHUB_TOKEN.
- Добавить в раздел [переменных GitHub репозитория](#) переменную DOCHUB_USERNAME с именем пользователя DockerHub. Эта переменная должна совпадать с одноимённым секретом. Такое дублирование нужно, т.к. GitHub заменяет значения секретов на звёздочки, при попытке преобразовать их в текст, а для указания полного имени Docker образа нам нужно имя пользователя. В принципе для имени пользователя можно ограничиться только переменной и не создавать для него секрет.

91. [docker] Закоммитьте созданный workflow, но пока не отправляйте на GitHub.

92. [docker] Добавим Dockerfile. Для этого в корне репозитория создайте каталог "docker" и в нём создайте файл с именем "Dockerfile" содержащий (версию python укажите свою):

```

FROM python:3.10.6-alpine
WORKDIR /app
COPY requirements.txt /app
COPY /src/app /app
RUN pip install -r requirements.txt

```



```
ENTRYPOINT [ "python" ]  
CMD [ "app.py" ]
```

```
(venv) kemran@kemran:~/var_keeper$ git add .  
(venv) kemran@kemran:~/var_keeper$ git commit -m "Сделал воркфлоу"  
[docker d764297] Сделал воркфлоу  
1 file changed, 37 insertions(+)  
create mode 100644 .github/workflows/staging.yml  
(venv) kemran@kemran:~/var_keeper$ mkdir docker  
(venv) kemran@kemran:~/var_keeper$ nano docker/Dockerfile
```

Это

почти тот же Dockerfile, который мы использовали ранее. Отличия только в команде COPY. Здесь, мы копируем "requirement.txt" из корня репозитория в каталог "/app" в контейнере, а затем копируем содержимое каталога "/src/app" из репозитория тоже в "/app" в контейнере.

93. [docker] Закоммитьте Dockerfile. На GitHub пока не отправляем.

```
(venv) kemran@kemran:~/var_keeper$ git add .  
(venv) kemran@kemran:~/var_keeper$ git commit -m "Make Dockerfile"  
[docker 32abeab] Make Dockerfile  
1 file changed, 7 insertions(+)  
create mode 100644 docker/Dockerfile  
(venv) kemran@kemran:~/var_keeper$
```

94. Откройте репозиторий на GitHub и в его настройках добавьте в раздел [секретов](#) (secrets) переменную DOCHUB_USERNAME - имя пользователя от аккаунта DockerHub и DOCHUB_TOKEN - пароль от него. Там же, но на соседней вкладке (variables) добавьте переменную DOCHUB_USERNAME с именем пользователя DockerHub.

Здесь мы используем в качестве репозитория для контейнера - DockerHub, но GitHub так же позволяет хранить Docker образы в своём реестре пакетов на ghcr.io (подробнее в [документации](#)).

The screenshot shows the GitHub repository settings page for a repository. The 'Secrets' tab is selected, and the 'Environment secrets' section is visible. Below it, the 'Repository secrets' section is partially visible. The 'Variables' tab is also visible, and the 'Repository variables' section is shown. A table of repository variables is displayed, with one variable named 'DOCHUB_USERNAME' having the value 'remr20052' and being updated '1 minute ago'.

Name	Value	Last updated
DOCHUB_USERNAME	remr20052	1 minute ago

95. [docker] Теперь можно отправить все созданные коммиты на GitHub.

```
(venv) kemran@kemran:~/var_keeper$ git push -u origin docker
Username for 'https://github.com': remr2005
Password for 'https://remr2005@github.com':
Перечисление объектов: 12, готово.
Подсчет объектов: 100% (12/12), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (7/7), готово.
Запись объектов: 100% (9/9), 1.38 КиБ | 1.38 МиБ/с, готово.
Всего 9 (изменений 2), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'docker' on GitHub by visiting:
remote:   https://github.com/remr2005/var_keeper/pull/new/docker
remote:
To https://github.com/remr2005/var_keeper
 * [new branch]      docker -> docker
Ветка "docker" создает новую ветку "docker" из "origin/master"
```

96. Создайте pull request и для слияния выберете ветки "dev" и "docker". Убедитесь, что тесты проходят успешно и завершите слияние. Удалите ветку "docker" из репозитория на GitHub.

Docker #2



Open

remr2005 wants to merge 2 commits into `dev` from `docker`



Conversation 0



Commits 2



Checks 1



Files changed 2



remr2005 commented now

Owner



No description provided.



remr2005 added 2 commits [19 minutes ago](#)



Сделал воркфлоу

d764297



Make Dockerfile

✓ 32abeab

Add more commits by pushing to the `docker` branch on [remr2005/var_keeper](#).



Require approval from specific reviewers before merging

[Rulesets](#) ensure specific people approve pull requests before they're merged.

Add rule



All checks have passed

2 successful checks

[Show all checks](#)



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



or view [command line instructions](#).



Add a comment

Write

Preview

H

B

I

≡

<>

🔗

≡

≡

≡

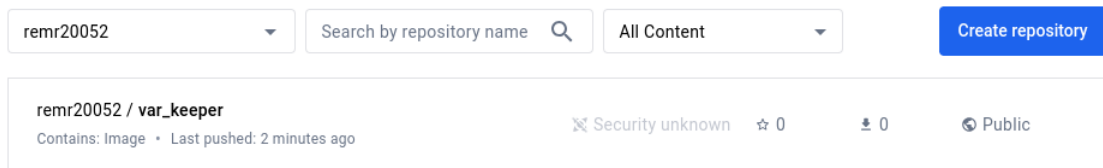
🔗

@

...

97. Перейдите в раздел "Actions" и убедитесь, что последний workflow выполнен успешно (это тот который собирает и отправляет контейнер на DockerHub). Затем перейдите в свой

репозиторий на DockerHub и убедитесь, что в списке тегов появился новый образ.



98. Проверьте работоспособность образа запустив его при помощи docker (не забудьте пробросить порт).

99. [docker -> dev] В терминале перейдите в ветку "dev" и заберите изменения с GitHub в локальный репозиторий: git pull, затем удалите локальную ветку "docker".

```
(venv) kemran@kemran:~/var_keeper$ git pull
Обновление 3b7cae2..cd4bb2d
Fast-forward
 .github/workflows/staging.yml | 37 +++++
 docker/Dockerfile             |  7 +
 2 files changed, 44 insertions(+)
 create mode 100644 .github/workflows/staging.yml
 create mode 100644 docker/Dockerfile
(venv) kemran@kemran:~/var_keeper$ git branch -d docker
Ветка docker удалена (была 32abeab).
(venv) kemran@kemran:~/var_keeper$
```

100. Для того, чтобы быстро получать информацию о состоянии последней stage версии и о коммите для которого построена эта версия добавим в "ReadMe.md" бейдж. По сути бейдж - это просто изображение, которое отображает некоторую информацию и автоматически изменяется, при изменении этой информации.

Бедж можно добавить 3 способами:

- Используя встроенные средства GitHub. Довольно ограниченный метод, позволяет создавать [бейджи статуса](#) показывающие успешность его выполнения указанного workflow.
- Используя внешние сервисы генерирующие изображения. Очень хороший вариант [shields.io](#). Данный сервис содержит кучу заготовок, которые можно настраивать по своему желанию.
- Использовать просто статическую картинку.

104. Для начала добавим бейдж отображающий статус процесса сборки и отправки образа на DockerHub. Файл описывающий соответствующий workflow называется у нас "staging.yml". Бейдж можно получить по ссылке в формате:

https://github.com/<OWNER>/<REPOSITORY>/actions/workflows/<WORKFLOW_FILE>/badge.svg

<OWNER> - это имя владельца репозитория; <REPOSITORY> - это название репозитория; <WORKFLOW_FILE> - это название файла рабочего процесса (с расширением!). В нашем случае staging.yml. Регистр букв - важен!

Второй вариант получения этого бейджа - через веб-интерфейс GitHub. Для этого в разделе Actions выберите интересующий workflow и затем в выпадающем меню "Create status badge":

```
(venv) kemran@kemran:~/var_keeper$ curl https://github.com/remr2005/var_keeper/actions/workflows/staging.yml/badge.svg
<svg xmlns="http://www.w3.org/2000/svg" width="263" height="20">
  <title>Push Stage version to DockerHub - passing</title>
  <defs>
    <linearGradient id="workflow-fill" x1="50%" y1="0%" x2="50%" y2="100%">
      <stop stop-color="#444D56" offset="0%"></stop>
      <stop stop-color="#24292E" offset="100%"></stop>
    </linearGradient>
    <linearGradient id="state-fill" x1="50%" y1="0%" x2="50%" y2="100%">
      <stop stop-color="#34D058" offset="0%"></stop>
      <stop stop-color="#28A745" offset="100%"></stop>
    </linearGradient>
  </defs>
  <g fill="none" fill-rule="evenodd">
    <g font-family="&#39;DejaVu Sans&#39;;Verdana,Geneva,sans-serif" font-size="11">
      <path id="workflow-bg" d="M0,3 C0,1.3431 1.3552,0 3.02702703,0 L213,0 L213,20 L3.02702703,20 C1.3552,20 0,18.656
9 0,17 L0,3 Z" fill="url(#workflow-fill)" fill-rule="nonzero"></path>
      <text fill="#010101" fill-opacity=".3">
        <tspan x="22.1981982" y="15" aria-hidden="true">Push Stage version to DockerHub</tspan>
      </text>
      <text fill="FFFFFF">
        <tspan x="22.1981982" y="14">Push Stage version to DockerHub</tspan>
      </text>
    </g>
    <g transform="translate(213)" font-family="&#39;DejaVu Sans&#39;;Verdana,Geneva,sans-serif" font-size="11">
      <path d="M0 0h46.939C48.629 0 50 1.343 50 3v14c0 1.657-1.37 3-3.061 3H0V0z" id="state-bg" fill="url(#state-fill)"
" fill-rule="nonzero"></path>
      <text fill="#010101" fill-opacity=".3" aria-hidden="true">
        <tspan x="4" y="15">passing</tspan>
      </text>
      <text fill="FFFFFF">
        <tspan x="4" y="14">passing</tspan>
      </text>
    </g>
    <path fill="#959DA5" d="M11 3c-3.868 0-7 3.132-7 7a6.996 6.996 0 0 0 4.786 6.641c.35.062.482-.148.482-.332 0-.166-
.01-.718-.01-1.304-1.758.324-2.213-.429-2.353-.822-.079-.202-.42-.823-.717-.99-.245-.13-.595-.454-.01-.463.552-.009.94
6.508 1.077.718.63 1.058 1.636.76 2.039.577.061-.455.245-.761.446-.936-1.557-.175-3.185-.779-3.185-3.456 0-.762.271-1.
392.718-1.882-.07-.175-.315-.892.07-1.855 0 0 .586-.183 1.925.718a6.5 6.5 0 0 1 1.75-.236 6.5 6.5 0 0 1 1.75.236c1.338
-.91 1.925-.718 1.925-.718.385.963.14 1.68.07 1.855.446.49.717 1.112.717 1.882 0 2.686-1.636 3.28-3.194 3.456.254.219.
473.639.473 1.295 0 .936-.009 1.689-.009 1.925 0 .184.131.402.481.332A7.011 7.011 0 0 0 18 10c0-3.867-3.133-7-7z"></
path>
  </g>
</svg>
```

105. [dev] По правилам, для добавления каждого **нового функционала** мы должны создать **новую ветку**, но для уменьшения размера работы выполним следующие несколько шагов сразу в ветке "dev".
- Разместим бейдж на главной странице нашего репозитория. Для этого откройте "ReadMe.md" и добавьте после заготовка ссылку на бейдж. Чтобы бейдж отображался как картинка можно воспользоваться как синтаксисом [html](#), так и [Markdown](#). В моём случае файл будет выглядеть так:

```
# Var_keeper
```

```
![]
```

```
(https://github.com/VladimirChabanov/var_keeper/actions/workflows/staging.yml/badge.svg)
```

Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.

```
GNU nano 6.2
```

```
ReadMe.md *
```

```
# Var_keeper
```

```

```

```
Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.
```

106. Второй бейдж добавим при помощи сервиса shields.io.
Перейдите по ссылке и введите в строке поиска: "docker". В списке выберите пункт "Docker Image Version". В форме справа щёлкните по пункту "Show optional parameters" и заполните поля: user repo sort и label. user - логин на DockerHub; repo = var_keeper; sort = date; label = build for commit. Остальное можете заполнить по желанию. Внизу страницы щелкните по кнопке "Copy", чтобы получить просто ссылку на изображение или сразу выберите в каком виде хотите её получить (HTML, markdown).

user — path

remr20052

repo — path

var_keeper

✕ Hide optional parameters

sort — query

date

arch — query

style — query

logo — query

One of the named logos (bitcoin, dependabot, gitlab, npm, payl

logoColor — query

The color of the logo (hex, rgb, rgba, hsl, hsla and css named c

logoSize — query

Make icons adaptively resize by setting auto . Useful for some \

label — query

build for commit

107. [dev] Добавьте бейдж в ReadMe.md через пробел после первого. В моём случае файл будет выглядеть так:

Var_keeper

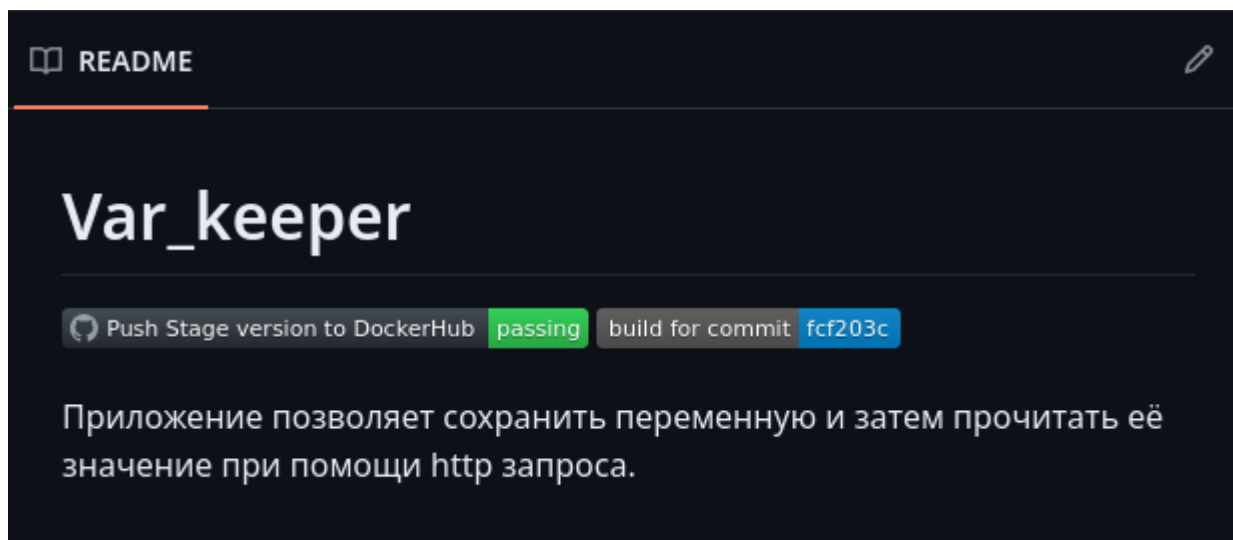
```
![]
(https://github.com/VladimirChabanov/var_keeper/actions/workflows/
staging.yml/badge.svg) ![]
(https://img.shields.io/docker/v/vladimirchabanov/var_keeper?
label=build%20for%20commit&sort=date)
```

```
)
```

Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.

108. [dev] Закоммитьте изменения, отправьте на GitHub и убедитесь, что бейджи отображаются нормально. Не забудьте выбрать ветку "dev" в интерфейсе GitHub, т.к. пока что "ReadMe.md" изменён только в ней.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Новый бэйдж"
[dev fcf203c] Новый бэйдж
1 file changed, 3 insertions(+)
(venv) kemran@kemran:~/var_keeper$ git push
Перечисление объектов: 5, готово.
Подсчет объектов: 100% (5/5), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (3/3), готово.
Запись объектов: 100% (3/3), 562 байта | 562.00 КиБ/с, готово.
Всего 3 (изменений 1), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/remr2005/var_keeper
cd4bb2d..fcf203c dev -> dev
(venv) kemran@kemran:~/var_keeper$
```



109. Осталось настроить последний шаг, слияние "dev" и "main", но перед этим добавим ещё одну имитацию наличия тестов. Будем считать, что перед вливанием в ветку "main" мы хотим протестировать код максимально полно, пусть это и займёт

много времени. Для этого, помимо обычных тестов, мы хотим запустим все остальные.

110. [dev] В каталоге "test" создайте файл "all-mega-hard-tests.sh" со следующим содержанием:

```
GNU nano 6.2 test/all-mega-hard-tests.sh
echo "All Mega Hard Tests PASS"
```

111. [dev] КОММИТ.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "мега тяжелые тесты"
[dev 8028dc7] мега тяжелые тесты
1 file changed, 1 insertion(+)
create mode 100644 test/all-mega-hard-tests.sh
```

112. [dev] Слияние "dev" и "main" тоже будем выполнять через pull request, в процессе которого будем запускать все тесты, которые у нас есть. Для этого в каталоге ".github/workflows/" создайте файл "pull-request-to-main.yml" содержащий:

name: Pull Request To Main

run-name: Run All Tests

on:

pull_request:

branches:

- 'main'

jobs:

unit_testing:

runs-on: ubuntu-latest

steps:

- name: Checkout

uses: actions/checkout@v3

- name: Run Unit Tests

run: |

chmod +x ./test/unit-tests.sh

./test/unit-tests.sh

integration_testing:

needs: unit_testing

runs-on: ubuntu-latest

steps:

- name: Checkout
uses: actions/checkout@v3
- name: Install requirements
run: pip install -r requirements.txt
- name: Run app
run: python3 ./src/app/app.py &
- name: Test GET-request
run: |
 ANSW=\$(curl http://127.0.0.1:5000/)
 if ["\$ANSW" != "Hello, World!"]; then
 exit 1
 else
 echo "Integration Test PASS"
 fi

mega_hard_testing:

needs: [unit_testing, integration_testing]

runs-on: ubuntu-latest

steps:

- name: Checkout
uses: actions/checkout@v3
- name: Run Mega Hard Tests
run: |
 chmod +x ./test/all-mega-hard-tests.sh
 ./test/all-mega-hard-tests.sh

```

GNU nano 6.2 .github/workflows/pull-request-to-main.yml
name: Pull Request To Main
run-name: Run All Tests
on:
  pull_request:
    branches:
      - 'main'
jobs:
  unit_testing:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Run Unit Tests
        run: |
          chmod +x ./test/unit-tests.sh
          ./test/unit-tests.sh

  integration_testing:
    needs: unit_testing
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Install requirements
        run: pip install -r requirements.txt
      - name: Run app
        run: python3 ./src/app/app.py &
      - name: Test GET-request
        run: |
          ANSW=$(curl http://127.0.0.1:5000/)
          if [ "$ANSW" != "Hello, World!" ]; then
            exit 1
          else
            echo "Integration Test PASS"
          fi

  mega_hard_testing:
    needs: [unit_testing, integration_testing]
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Run Mega Hard Tests
        run: |
          chmod +x ./test/all-mega-hard-tests.sh
          ./test/all-mega-hard-tests.sh

```

113. [dev] КОММИТ и пуш на GitHub.

```

(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Pull реквест на мейн"
[dev c126c01] Pull реквест на мейн
1 file changed, 46 insertions(+)
create mode 100644 .github/workflows/pull-request-to-main.yml

```

```
(venv) kemran@kemran:~/var_keeper$ git push
Перечисление объектов: 13, готово.
Подсчет объектов: 100% (13/13), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (7/7), готово.
Запись объектов: 100% (9/9), 1.19 КиБ | 1.19 МиБ/с, готово.
Всего 9 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To https://github.com/remr2005/var_keeper
   fcf203c..c126c01  dev -> dev
```

114. Перейдите на GitHub и выполните pull request. В настройках выберите для слияния ветки "main" и "dev". Под списком коммитов вы должны увидеть, что были запущены unit, intergation и mega hard тесты (возможно придётся подождать несколько секунд пока они запустятся). Завершите слияние.

The screenshot shows a GitHub pull request titled "Dev #3" where user "remr2005" wants to merge 12 commits into the "main" branch from the "dev" branch. The commit history includes:

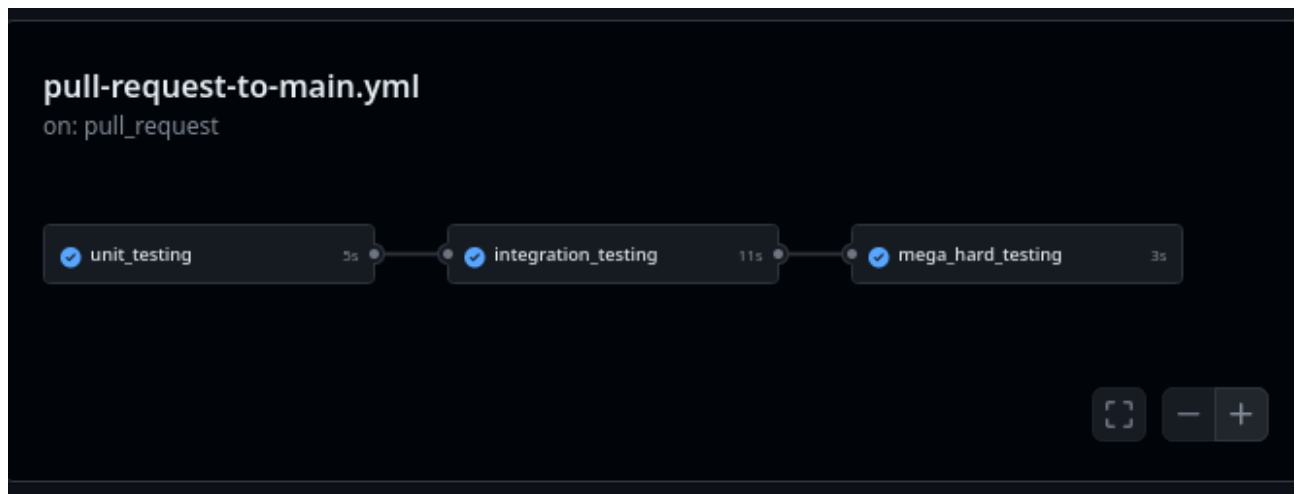
- remr2005 and others added 12 commits 3 days ago
- Добавлен шаблон приложения (8adff81)
- Добавлены unit тесты (cc9d5d5)
- s (cc6c040)
- Fix mistake in unit-tests (2a1d6fb)
- Pull request to dev (a6c6cf8)
- Merge pull request #1 from remr2005/template (Verified, 3b7cae2)
- Сделал воркфлоу (d764297)
- Make Dockerfile (32abeab)
- Merge pull request #2 from remr2005/docker (Verified, cd4bb2d)
- Новый бэйдж (fcf203c)
- мега тяжелые тесты (8028dc7)
- Pull реквест на мейн (c126c01)

Below the commit history, there is a section for "Add more commits by pushing to the dev branch on remr2005/var_keeper." and a summary of checks:

- Require approval from specific reviewers before merging** (Add rule button)
- All checks have passed** (4 successful checks, Show all checks button)
- This branch has no conflicts with the base branch** (Merging can be performed automatically.)

At the bottom, there is a "Merge pull request" button and a link to "or view command line instructions."

115. Перейдите в раздел "Actions" и в запущенных workflows выберите последний (Run All Tests). Вы увидите, что job "unit_testing", "integration_testing" и "mega_hard_testing" объединены в цепочку.



116. Откройте репозиторий GitHub, выберите ветку "main" и убедитесь, что "ReadMe.md" соответствует тому, что было в ветке "dev" до слияния.

main

Go to file

+

<> Code

remr2005

Merge pull request #3 from r...

d86fa6f · 1 minute ago

15 Commits

.github/workflows	Pull реквест на мейн	5 minutes ago
docker	Make Dockerfile	3 hours ago
src/app	Добавлен шаблон приложения	3 days ago
test	мега тяжелые тесты	7 minutes ago
.gitignore	Добавлен .gitignore	4 days ago
ReadMe.md	Новый бэйдж	12 minutes ago
requirements.txt	Добавлен шаблон приложения	3 days ago

README

Var_keeper

Push Stage version to DockerHub

passing

build for commit

c126c01

Приложение позволяет сохранить переменную и затем прочитать её значение при помощи http запроса.

117. [dev -> main] Перед тем, как продолжить не забудьте забрать все изменения с GitHub в локальный репозиторий (git pull в ветке "main").

```

(venv) kemran@kemran:~/var_keeper$ git pull
Обновление a0c9b30..d86fa6f
Fast-forward
 .github/workflows/pull-request-to-dev.yml | 24 ++++++
 .github/workflows/pull-request-to-main.yml | 46 ++++++
 .github/workflows/staging.yml             | 37 ++++++
 .github/workflows/unit_tests.yml          | 18 ++++++
 ReadMe.md                                | 3 +++
 docker/Dockerfile                         | 7 ++++++
 requirements.txt                          | 7 ++++++
 src/app/app.py                            | 10 ++++++
 test/all-mega-hard-tests.sh               | 1 +
 test/unit-tests.sh                       | 1 +
10 files changed, 154 insertions(+)
create mode 100644 .github/workflows/pull-request-to-dev.yml
create mode 100644 .github/workflows/pull-request-to-main.yml
create mode 100644 .github/workflows/staging.yml
create mode 100644 .github/workflows/unit_tests.yml
create mode 100644 docker/Dockerfile
create mode 100644 requirements.txt
create mode 100644 src/app/app.py
create mode 100644 test/all-mega-hard-tests.sh
create mode 100644 test/unit-tests.sh
(venv) kemran@kemran:~/var_keeper$

```

118. Чтобы проверить, что всё работает правильно, модифицируем проект.

119. [main -> dev] Для начала перейдите в "dev" и создайте новую ветку с именем "add_data_base". Все новые ветки должны создаваться от "dev"!

```

(venv) kemran@kemran:~/var_keeper$ git branch add_data_base
(venv) kemran@kemran:~/var_keeper$

```

120. [dev -> add_data_base] Перейдите в ветку "add_data_base". Тут будем вести разработку.

```

(venv) kemran@kemran:~/var_keeper$ git checkout add_data_base
Переключились на ветку «add_data_base»

```

121. Модифицируйте код приложения "app.py" (src/app/) следующим образом:

```

from flask import Flask, request
from getpass import getpass
from mysql.connector import connect, Error

connection = None

def init_db():

```

```

global connection
try:
    print('Connection to db:', end='')
    connection = connect(host='db', user='root',
password='123')
    print('OK')

    print('Create db:', end='')
    create_db_query = "CREATE DATABASE IF NOT EXISTS vars"
    with connection.cursor() as cursor:
        cursor.execute(create_db_query)
    print('OK')

    print('Change db:', end='')
    use_db_query = "USE vars"
    with connection.cursor() as cursor:
        cursor.execute(use_db_query)
    print('OK')

    print('Create table:', end='')
    create_table_query = """
CREATE TABLE IF NOT EXISTS vars(
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    value VARCHAR(100),
    UNIQUE (name)
)
"""
    with connection.cursor() as cursor:
        cursor.execute(create_table_query)
        connection.commit()
    print('OK')

```



```

except Error as e:
    print('Failure', e)

app = Flask(__name__)

@app.route('/var/<var_name>', methods=['GET'])
def get(var_name):
    select_query = f"""
    SELECT value FROM vars
    WHERE name = '{var_name}'
    """

    print("Select query:", select_query)
    with connection.cursor() as cursor:
        cursor.execute(select_query)
        return cursor.fetchall()[0][0]

@app.route('/var/<var_name>', methods=['POST'])
def set(var_name):
    value = request.form.get("value")
    insert_query = f"""
    INSERT INTO vars (name, value)
    VALUES ('{var_name}', '{value}')
    ON DUPLICATE KEY UPDATE value='{value}'
    """

    print("Insert query:", insert_query)
    with connection.cursor() as cursor:
        cursor.execute(insert_query)
        connection.commit()

```

```
return 'OK'
```

```
if __name__ == "__main__":
```

```
    init_db()
```

```
    app.run(debug=True, host='0.0.0.0')
```

```
GNU nano 6.2 src/app/app.py
from flask import Flask, request
from getpass import getpass
from mysql.connector import connect, Error

connection = None

def init_db():
    global connection
    try:
        print('Connection to db:', end='')
        connection = connect(host='db', user='root', password='123')
        print('OK')

        print('Create db:', end='')
        create_db_query = "CREATE DATABASE IF NOT EXISTS vars"
        with connection.cursor() as cursor:
            cursor.execute(create_db_query)
        print('OK')

        print('Change db:', end='')
        use_db_query = "USE vars"
        with connection.cursor() as cursor:
            cursor.execute(use_db_query)
        print('OK')

        print('Create table:', end='')
        create_table_query = """
        CREATE TABLE IF NOT EXISTS vars(
            id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(100),
            value VARCHAR(100),
            UNIQUE (name)
        )
        """
        with connection.cursor() as cursor:
            cursor.execute(create_table_query)
            connection.commit()
        print('OK')
    except Error as e:
        print('Failure', e)

app = Flask(__name__)

@app.route('/var/<var_name>', methods=['GET'])
def get(var_name):
    select_query = f"""
    SELECT value FROM vars
    WHERE name = '{var_name}'
    """
    print("Select query:", select_query)
```

122. [add_data_base] КОММИТ.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Change app.py"
[add_data_base f85c722] Change app.py
1 file changed, 77 insertions(+), 10 deletions(-)
rewrite src/app/app.py (72%)
(venv) kemran@kemran:~/var_keeper$
```

123. [add_data_base] Приложение теперь взаимодействует с базой данных mysql и для его работы нужно установить дополнительный пакет: mysql-connector-python.

Если вы уже деактивировали виртуальное окружение "venv", активируйте его снова, затем установите требуемый пакет: `pip install mysql-connector-python` и обновите "requirements.txt": `pip freeze > requirements.txt`

```
(venv) kemran@kemran:~/var_keeper$ pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.4.0-cp310-cp310-manylinux_2_17_x86_64.whl (19.4 MB)
    19.4/19.4 MB 15.4 MB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-8.4.0
(venv) kemran@kemran:~/var_keeper$ pip freeze > requirements.txt
```

124. [add_data_base] КОММИТ.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Update requirements"
[add_data_base 9c51cea] Update requirements
1 file changed, 1 insertion(+)
```

125. [add_data_base] Т.к. приложение изменилось и следовательно оно уже не будет проходить intergation тест. Поправим и его. Модифицируйте "pull-request-to-dev.yml" (.github/workflows/) следующим образом:

```
name: Pull Request To Dev
run-name: Run Integration Tests
on:
  pull_request:
    branches:
      - 'dev'
jobs:
  integration_testing:
    runs-on: ubuntu-latest
    container: python:3.10.9-slim
    services:
      db:
        image: mysql
        env:
          MYSQL_ROOT_PASSWORD: 123
        options: >-
          --health-cmd "mysqladmin ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Install requirements
        run: pip install -r requirements.txt
      - name: Run app
        run: python3 ./src/app/app.py &
      - name: Install curl
        run: apt-get -y update; apt-get -y install curl
      - name: Set variable a in 123
```

```

run: curl -X POST -F 'value=123' http://localhost:5000/var/a
- name: Test getting the value of a variable
  run: |
    VALUE=$(curl http://localhost:5000/var/a)
    if [ "$VALUE" != "123" ]; then
      exit 1
    else
      echo "Integration Test PASS"
    fi

```

Здесь весь `job` "integration_testing" запускается в `docker` контейнере "python:3.10.9-slim" (раздел `container`). До этого момента все `workflow` выполнялись на самой виртуальной машине, без контейнеризации. Кроме того, перед тем, как начнут выполняться действия указанные в разделе `steps` будет запущен `docker` контейнер с базой данных "mysql" (раздел `services`). Гарантируется, что все сервисы будут запущены до того, как начнётся исполнения `job`-а. Но запуск сервиса - это просто факт старта его `docker` контейнера, поэтому в разделе `options` сервиса `db` указан набор дополнительных команд (health check), которые пингуют готовность базы данных с заданным интервалом. Без этих команд исполнение `steps` может начаться раньше, чем база данных будет готова принять подключение.

По умолчанию все контейнеры запущенные в `job` подключены к одной сети, поэтому нет необходимости пробрасывать порты. Если бы мы не поместили весь `job` в `docker` контейнер, пришлось бы пробросить порт базы данных и менять в скрипте "app.py" `connect(host='db', user='root', password='123')` на `connect(host='localhost', user='root', password='123')`.

```

GNU nano 6.2 .github/workflows/pull-request-to-dev.yml
name: Pull Request To Dev
run-name: Run Integration Tests
on:
  pull_request:
    branches:
      - 'dev'
jobs:
  integration_testing:
    runs-on: ubuntu-latest
    container: python:3.10.9-slim
    services:
      db:
        image: mysql
        env:
          MYSQL_ROOT_PASSWORD: 123
        options: >-
          --health-cmd "mysqladmin ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Install requirements
        run: pip install -r requirements.txt
      - name: Run app
        run: python3 ./src/app/app.py &
      - name: Install curl
        run: apt-get -y update; apt-get -y install curl
      - name: Set variable a in 123
        run: curl -X POST -F 'value=123' http://localhost:5000/var/a
      - name: Test getting the value of a variable
        run: |
          VALUE=$(curl http://localhost:5000/var/a)
          if [ "$VALUE" != "123" ]; then
            exit 1
          else
            echo "Integration Test PASS"
          fi

```

126. [add_data_base] КОММИТ.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Update pull-request"
[add_data_base f7c59e9] Update pull-request
1 file changed, 18 insertions(+), 3 deletions(-)
```

127. [add_data_base] Данный тест у нас присутствует и в файле "pull-request-to-main.yml". Замените в нём содержимое job-а "integration_testing" на новое (из файла выше).

```
GNU nano 6.2 .github/workflows/pull-request-to-main.yml *
name: Pull Request To Main
run-name: Run All Tests
on:
  pull_request:
    branches:
      - 'main'
jobs:
  unit_testing:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Run Unit Tests
        run: |
          chmod +x ./test/unit-tests.sh
          ./test/unit-tests.sh

  integration_testing:
    runs-on: ubuntu-latest
    container: python:3.10.9-slim
    services:
      db:
        image: mysql
        env:
          MYSQL_ROOT_PASSWORD: 123
        options: >-
          --health-cmd "mysqladmin ping"
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5

  mega_hard_testing:
    needs: [unit_testing, integration_testing]
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
      - name: Run Mega Hard Tests
        run: |
          chmod +x ./test/all-mega-hard-tests.sh
          ./test/all-mega-hard-tests.sh
```

128. [add_data_base] КОММИТ и пуш на GitHub.

```
(venv) kemran@kemran:~/var_keeper$ git add .
(venv) kemran@kemran:~/var_keeper$ git commit -m "Change integration test"
[add_data_base c3528e1] Change integration test
 1 file changed, 11 insertions(+), 16 deletions(-)
(venv) kemran@kemran:~/var_keeper$ git push -u origin add_data_base
Перечисление объектов: 27, готово.
Подсчет объектов: 100% (27/27), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (14/14), готово.
Запись объектов: 100% (18/18), 2.98 КиБ | 1.49 МиБ/с, готово.
Всего 18 (изменений 5), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (5/5), completed with 1 local object.
remote:
remote: Create a pull request for 'add_data_base' on GitHub by visiting:
remote:   https://github.com/remr2005/var_keeper/pull/new/add_data_base
remote:
To https://github.com/remr2005/var_keeper
 * [new branch]      add_data_base -> add_data_base
Ветка «add_data_base» отслеживает внешнюю ветку «add_data_base» из «origin».
```

129. Перейдите на GitHub и выполните pull request. В настройках выберите для слияния ветки "dev" и "add_data_base". Дождитесь, пока все тесты выполнятся (успешно!) и подтвердите слияние. После слияния удалите ветку add_data_base.

Open

Add data base #4

remr2005 wants to merge 4 commits into `dev` from `add_data_base`

remr2005 commented 1 minute ago

Owner

...

No description provided.

remr2005 added 4 commits 13 minutes ago

Change `app.py`

f85c722

Update `requirements`

9c51cea

Update `pull-request`

f7c59e9

Change `integration test`

✓ c3528e1

Add more commits by pushing to the `add_data_base` branch on `remr2005/var_keeper`.

Require approval from specific reviewers before merging

[Rulesets](#) ensure specific people approve pull requests before they're merged.

Add rule

×

✓

All checks have passed

2 successful checks

Show all checks

✓

This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request

▼

or view [command line instructions](#).

130. Выполните pull request. В настройках выберите для слияния ветки "main" и "dev". Дождитесь, пока все тесты выполняются (успешно!) и подтвердите слияние.

Dev #5



remr2005 wants to merge 5 commits into `main` from `dev`



Conversation 0



Commits 5



Checks 1



Files changed 4



remr2005 commented now

Owner



No description provided.



remr2005 and others added 5 commits [16 minutes ago](#)



Change `app.py`

f85c722



Update `requirements`

9c51cea



Update `pull-request`

f7c59e9



Change `integration test`

✓ c3528e1



Merge pull request [#4](#) from remr2005/add_data_base

Verified

✓ 53090f6



Add more commits by pushing to the `dev` branch on [remr2005/var_keeper](#).



Require approval from specific reviewers
before merging

Add rule



[Rulesets](#) ensure specific people approve pull requests before they're merged.



All checks have passed

Show all checks

1 successful check



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



or view [command line instructions](#).

131. [add_data_base -> main] Перейдите в ветку "main" и заберите с GitHub изменения в локальный репозиторий.

```
(venv) kemran@kemran:~/var_keeper$ git checkout main
ПереклЮчилиcь на ветку «main»
Эта ветка соответствует «origin/main».
(venv) kemran@kemran:~/var_keeper$ git pull
remote: Enumerating objects: 2, done.
remote: Counting objects: 100% (2/2), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 0), reused 0 (delta 0), pack-reused 0
Распаковка объектов: 100% (2/2), 1.73 КиБ | 1.73 МиБ/с, готово.
Из https://github.com/remr2005/var_keeper
   d86fa6f..9c15262  main       -> origin/main
   c126c01..53090f6  dev        -> origin/dev
Обновление d86fa6f..9c15262
Fast-forward
 .github/workflows/pull-request-to-dev.yml | 21 ++++++++-----
 .github/workflows/pull-request-to-main.yml | 27 ++++++++-----
 requirements.txt                          |  1 +
 src/app/app.py                            | 83 ++++++++-----
 4 files changed, 105 insertions(+), 27 deletions(-)
(venv) kemran@kemran:~/var_keeper$
```

132. [main -> dev] Перейдите в ветку "dev" и заберите с GitHub изменения в локальный репозиторий.

```
(venv) kemran@kemran:~/var_keeper$ git checkout dev
ПереклЮчилиcь на ветку «dev»
Ветка отстает от «origin/dev» на 5 коммитов и может быть быстро перемотана.
(используйте «git pull», чтобы обновить вашу локальную ветку)
(venv) kemran@kemran:~/var_keeper$ git pull
Обновление c126c01..53090f6
Fast-forward
 .github/workflows/pull-request-to-dev.yml | 21 ++++++++-----
 .github/workflows/pull-request-to-main.yml | 27 ++++++++-----
 requirements.txt                          |  1 +
 src/app/app.py                            | 83 ++++++++-----
 4 files changed, 105 insertions(+), 27 deletions(-)
(venv) kemran@kemran:~/var_keeper$
```

133. [dev] Удалите ветку "add_data_base", а затем посмотрите историю репозитория: git log --oneline --graph.

```
(venv) kemran@kemran:~/var_keeper$ git branch -d add_data_base
Ветка add_data_base удалена (была c3528e1).
(venv) kemran@kemran:~/var_keeper$
```

```
(venv) kemran@kemran:~/var_keeper$ git log --oneline --graph
* 53090f6 (HEAD -> dev, origin/dev) Merge pull request #4 from remr2005/add_data_base
|
| * c3528e1 (origin/add_data_base) Change integration test
| * f7c59e9 Update pull-request
| * 9c51cea Update requirements
| * f85c722 Change app.py
|/
* c126c01 Pull реквест на мейн
* 8028dc7 мега тяжелые тесты
* fcf203c Новый бэйдж
* cd4bb2d Merge pull request #2 from remr2005/docker
|
| * 32abeab (origin/docker) Make Dockerfile
| * d764297 Сделал воркфлоу
|/
* 3b7cae2 Merge pull request #1 from remr2005/template
|
| * a6c6cf8 Pull request to dev
| * 2a1d6fb Fix mistake in unit-tests
| * cc6c040 s
| * cc9d5d5 Добавлены unit тесты
| * 8adff81 Добавлен шаблон приложения
|/
* a0c9b30 Добавлен .gitignore
* 00f763a Добавлен ReadMe.md
(venv) kemran@kemran:~/var_keeper$
```

Часть II: GitLab CI/CD

Ответы на вопросы.

1 События триггеры для рабочего процесса:

- **push**: Каждый раз, когда кто-то делает push в репозиторий.
- **pull_request**: При создании, изменении или закрытии pull request.
- **schedule**: Запуск работы по расписанию, используя cron синтаксис.
- **workflow_dispatch**: Ручной запуск работы из GitHub.

Эти события указываются в разделе **on** в файле **.yaml** рабочего процесса.

2. Хранение секретной информации:

Секреты следует хранить в разделе **Secrets** настроек репозитория на GitHub. Они безопасны, так как зашифрованы и доступны в рабочих процессах через переменные окружения.

3. Получение почты владельца репозитория:

Да, можно получить почту владельца репозитория во время выполнения рабочего процесса, используя GitHub API для получения данных пользователя.

4. Спецификация виртуальных машин:

- **Windows**: 2 ядра CPU, 7 ГБ RAM, 14 ГБ SSD диска.
- **Linux**: 2 ядра CPU, 7 ГБ RAM, 14 ГБ SSD диска.

Эти характеристики могут быть обновлены, поэтому рекомендуется проверять актуальную документацию GitHub.

5. Список установленного ПО:

Список установленного ПО на виртуальные машины можно найти в официальной документации GitHub Actions, в разделе "Software installed on GitHub-hosted runners".

Вывод:

Я ознакомился с базовыми возможностями утилиты git, сервисов GitHub, GitLab и возможностей CI/CD, которые они предоставляют.