Портал с документацией своими руками

без devops'ов и разработчиков

Оглавление

Предварительная подготовка

1. Регистрируемся на GitHub.
2. Создаем свой первый проект. Это можно сделать сразу после регистрации или создать позже.
3. Устанавливаем Git: https://git-scm.com/download/win.
4. Устанавливаем Python, например последнюю версию: https://www.python.org/downloads/release/python-3121/. Пролистайте в самый низ и выберите версию для вашей ОС. Для пользователей MacOS, можно скачать все следующей командой:

```
brew install python
```

5. Для самых стойких — установить Docker: https://www.docker.com/get-started/. С помощью докера мы будем автоматизировать выкладку обновлений контента.
6. Перезапустите компьютер после установки всех компонент.
7. Проверьте, что Git и Python успешно установились:

```
git --version
git version 2.40.1.windows.1

python --version
Python 3.11.1

pip --version
pip 22.3.1 from C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip
(python 3.11)
```

Первые шаги

Мы будем использовать язык разметки Markdown и движок MkDocs для генерации статического контента.

1. Открываем терминал.
2. Устанавливаем MkDocs локально:

```
pip install mkdocs
```

3. Проверяем, что все установилось:

```
mkdocs --version
mkdocs, version 1.5.3 from C:\Users\User\AppData\Local\Programs\Python\Python311\Lib\site-
packages\mkdocs (Python 3.11)
```

4. Создаем свой первый проект в текущей директории:

```
mkdocs new my-project
```

5. Переходим в директорию с проектом:

```
cd my-project
```

6. Собираем и запускаем локально портал:

```
mkdocs serve
```

   В конце, в результате выполнения команды можно получить ссылку по которой доступен результат сборки портала:

```
INFO    -  Building documentation...
INFO    -  Cleaning site directory
INFO    -  Documentation built in 0.19 seconds
INFO    -  [16:00:36] Watching paths for changes: 'docs', 'mkdocs.yml'
INFO    -  [16:00:36] Serving on http://127.0.0.1:8000/
```

7. Копируем http://127.0.0.1:8000/ и вставляем в адресную строку.

Наводим красоту

Практически все визуальные настройки выполняются в рамках файла mkdocs.yml. Изначально он состоит лишь из одной строчки:

```
site_name: My Docs
```

Для MkDocs есть множество различных тем, плагинов и расширений. Самая популярная тема для MkDocs с активной поддержкой и постоянным развитием — это MkDocs Material: https://squidfunk.github.io/mkdocs-material/. Сделаем наш портал в такой же теме.

1. Установим тему Material:

```
pip install mkdocs-material
```

2. Устанавливаем пакет расширений (понадобится нам в будущем):

```
pip install pymdown-extensions
```

3. Откроем файл mkdocs.yml и подключим тему Material к нашему проекту:

```
theme:
    name: material # https://squidfunk.github.io/mkdocs-material/
```

4. Пересоберем портал:

```
mkdocs serve
```

5. Нам нужно добавить новый документ на наш портал и при это указать его в навигационном меню. Для этого в файле mkdocs.yml нужно создать новую секцию:

```
nav:
```

6. Теперь скачайте из Телеграмма файлик `my-first-docs-portal.md` . И перенесите его в папку **docs** в вашем проекте.
7. Добавим инструкцию на портал:

```
nav:
    - Как запустить первый портал?: my-first-docs-portal.md
```

8. Добавим расширение, которое сделает блоки кода более красивыми:

```
markdown_extensions:
    - pymdownx.superfences # более гибкое оформление блоков https://facelessuser.github.io/
pymdown-extensions/extensions/superfences/
```

9. Добавим группирующий раздел и перенесем навигационную панель наверх.

```
nav:
    - Как запустить первый портал?:
        - Вот так: my-first-portal.md
    - Дом: index.md
theme:
    name: material # основная тема, которую используем, https://squidfunk.github.io/mkdocs-
material/
    features:
        - navigation.tabs # перемещает навигационное меню наверх
```

10. Заменим логотип и favicon на нашем портале:

```
theme:
    name: material #  https://squidfunk.github.io/mkdocs-material/
    logo: TWD_logo.png
    favicon: TWD_favicon.png
```

11. Поменяем цвета портала на корпоративные. Для этого нужно внутри папки docs нужно создать файл material-styles.css и указать путь до него в mkdocs.yml:

```
extra_css:
    - material-styles.css
```

    А внутри файла material-styles.css укажем стили для шапки нашего портала:

```
.md-header {
--md-primary-fg-color: #FFFFFF;
```

```
--md-primary-bg-color: #000000;
}
```

12. Добавим поисковую строку на портал:

```
plugins:
    - search:
        lang:
          - ru
```

Итоговый `mkdocs.yml`:

```
site_name: Tech Writer Days
nav:
    - Как запустить первый портал?:
        - Вот так: my-first-portal.md
    - Дом: index.md
theme:
    name: material
    logo: TWD_logo.png
    favicon: TWD_favicon.png
    features:
        - navigation.tabs
extra_css:
    - material-styles.css
markdown_extensions:
    - pymdownx.superfences
plugins:
    - search:
        lang:
          - ru
```

Рекомендуемые инструменты:

1. Плагин для работы с видео на портале:

```
plugins:
    - mkdocs-video:
        is_video: True #изменение тега для видео на конечной странице html (было <iframe>,
стало <video>, когда true)
        video_type: mpeg #- если формат видео не mp4 (по умолчанию), а другой. Этот
параметр будет работать только с <video> тегом ( is_video: True)
        video_autoplay: True # автовоспроизведение видео. Этот параметр будет работать
только с <video> тегом ( is_video: True)
        video_loop: False # зацикливание видео. Этот параметр будет работать только с
<video> тегом ( is_video: True)
        video_muted: True # должно ли видео быть на мьюте. Этот параметр будет работать
только с <video> тегом ( is_video: True)
        video_controls: True # отображение элементов управления видео. Этот параметр будет
работать только с <video> тегом ( is_video: True)
        css_style:
          width: "100%" #изменение ширины видео по дефолту
```

2. Расширение, которое делает красивые примечание:

```
markdown_extensions:
    - admonition # https://squidfunk.github.io/mkdocs-material/setup/extensions/python-markdown/#admonition
```

> **Внимание**
>
> Вот например красивое внимание :)

3. Расширение, которое позволяет переиспользовать одинаковые части документации:

```
markdown_extensions:
    - pymdownx.snippets # Расширение Snippets добавляет возможность встраивать в документ содержимое из произвольных файлов, включая другие документы или исходные файлы
```

> **Примечание**
>
> Нам в последнее время стало более актуально использовать шаблонизатор Jinja, который работает в рамках плагина mkdocs-macros-plugin.

4. Очень полезным может быть плагин делающий редиректы: mkdocs-redirects.

```
plugins:
    - redirects:
        redirect_maps:
          'Support/new.md': 'LegalInformation/general_info.md'
```

Начинаем работать с Git

Чтобы сохранять, версионировать и запускать портал нам понадобится GitLab/GitHub. Самый эффективный и простой способ склонировать репозиторий без ограничений на push, это создать Personal Access Token и использовать его при клонировании репозитория:

1. В правом верхнем углу кликаем на аватар своего профиля.
2. В открывшемся меню выбираем раздел **Settings**.
3. В самом низу левого меню ищем раздел **Developer Settings** и кликаем на него.
4. Переходим в **Personal Access Tokens → Tokens (classic)**.
5. Кликаем на выпадающее меню **Generate new token** и выбираем **Generate new token (classic)**.
6. Задаем краткое описание токену и проставляем все галочки снизу.
7. В самом низу кликаем кнопку **Generate Token**.

Теперь наконец склонируем репозиторий, который мы создавали на шаге Предварительная подготовка.

1. Склонируем репозиторий в любое удобное место, но не в папку с нашим порталом.

```
PS C:\Users\User\Desktop> git clone https://github.com/Recours/recours.github.io.git
```

или через Personal Access Token:

```
git clone https://oauth2:{сгенерированный токен}@{ссылка до репозитория без https://}
```

Например:

```
git clone https://oauth2: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@github.com/diy-portal/
one-more-test
```

2. В моем случае, я клонировал на рабочий стол и у меня появилась папка с названием репозитория. Теперь нужно перенести все содержимое папки **my-project** внутрь новой папки.
3. Перейдем внутрь папки с названием репозитория.

```
cd recours.github.io
```

4. Собираем наш первый коммит:

```
git add . --all
git commit -m "Наш первый портал"
```

Итог:

```
[main 88fd875] Наш первый портал
 5 files changed, 206 insertions(+)
 create mode 100644 docs/VK_WorkSpace_logo.svg
 create mode 100644 docs/index.md
 create mode 100644 docs/material-styles.css
 create mode 100644 docs/my-first-docs-portal.md
 create mode 100644 mkdocs.yml
```

5. Отправим изменения в репозиторий на сервер GitHub'a:

```
git push -uf origin main
```

где `main` это название ветки.

Итог:

```
Перечисление объектов: 12, готово.
Подсчет объектов: 100% (12/12), готово.
При сжатии изменений используется до 8 потоков
Сжатие объектов: 100% (10/10), готово.
Запись объектов: 100% (10/10), 11.02 КиБ | 11.02 МиБ/с, готово.
Всего 10 (изменений 0), повторно использовано 0 (изменений 0), повторно использовано
пакетов 0
To https://github.com/Recours/DocsPortal.git
   e7a1c28..c9db308  main -> main
branch 'main' set up to track 'origin/main'.
```

Если вы как и я не фанаты работать с консолью, то рекомендую воспользоваться клиентами с полноценным UI для работы с Git:

- Sourcetree
- Fork
- GitHub Desktop

Публикуем контент в GitHub Pages

Чтобы опубликовать контент в GitHub Pages, нам нужно сделать следующие действия:

1. Переименовать репозиторий в виде: {GitHub-nickname}.github.io. Например: recours.github.io. Переименовать репозиторий можно в разделе Settings проекта. Это требование GitHub'a, иначе публикации не будет.
2. В том же разделе Settings, в левой боковой панели нам нужно перейти в раздел **Actions** → **General**.
3. Пролистать до подраздела **Workflow permissions** и выбрать радио-баттон с опцией «Read and write permissions».

Затем добавляем скрипт автоматизирующий выкладку контента на наш веб-портал:

1. В корневой папке нашего проекта создайте папку **.github**.
2. В папке **.github** создайте папку **workflows**.
3. В папке **workflows** создайте файл `mkdocs.yml`.
4. Вставьте следующий код в этот файл:

```
name: mkdocs
on:
  push:
    branches:
      - main
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-python@v2
        with:
          python-version: 3.x
      - run: pip install mkdocs-material
      - run: mkdocs gh-deploy --force
```

Что есть в этом скрипте:

- `name: mkdocs:` — это имя рабочего процесса.
- `on:` — определяет условия, при которых запускается данный рабочий процесс. В данном случае, рабочий процесс запускается при каждом пуше в ветку "main".

- `jobs:` — определяет список задач, которые должны быть выполнены в рамках этого рабочего процесса. `deploy` — название задачи, в рамках которой мы будем деплоить портал.
- `- uses: actions/checkout@v2` — клонирует репозиторий в рабочее пространство, где будет выполняться генерация статического контента и сборка портала.

5. Запушьте изменения в удаленный репозиторий:

```
git add . --all
git commit -m "Автоматическая публикация контента"
git push -uf origin main
```

После того, как все job'ы выполнятся нам нужно переключить ветку, с которой будет происходить публикация контента. Сейчас на {GitHub-nickname}.github.io скорее всего весит 404 ошибка. Поэтому мы:

6. Переходим в **Settings** репозитория.
7. В левой боковой панели выбираем раздел **Pages**.
8. В подразделе **Build and deployment** в дропдауне опции **Branch** выбираем ветку `gh-pages`.
9. Ждем чуть меньше минуты и снова заходим на портал.

Публикуем контент в GitLab Pages

Все очень похоже при публикации в GitLab Pages:

1. В корне проекта нужно создать файл `.gitlab-ci.yml`.
2. Вставить следующий код в файл с попракой на название веток в вашем репозитории:

```
# Используем легковесный образ чтобы минимизировать задержки на скачивание и установке образа
image: python:3-alpine

.run_only_on_master:
  &run_only_on_master
  rules:
    - if: '$CI_COMMIT_BRANCH == "main"'

build_static_site:
  stage: build
  image: python:3-alpine
  before_script:
    - pip install "Cython<3.0" pyyaml --user --no-build-isolation # гитхабные тикеты про багу
https://github.com/yaml/pyyaml/issues/601 и https://github.com/yaml/pyyaml/pull/702
    - pip install mkdocs
    - pip install mkdocs-material
    - pip install pymdown-extensions
  script:
    - mkdocs build -d public
  artifacts:
    expire_in: 15 mins
    paths:
      - public

# специальная джоба для выкладки в gitlab pages (для ее выполнения обязательна выполненная
```

```
джоба build_static_site)
pages:
  <<: *run_only_on_master
  stage: deploy
  needs:
    - build_static_site
  script: # это костыль для обхода известной баги гитлаба - нельзя джобу без скрипта. Поэтому
тут просто зовем команду-заглушку, которая ничего не делает
    - "true"
  artifacts:
    expire_in: 15 mins
    paths:
      - public
```

Как подключить локальный раннер к репозиторию в GitLab

Раннер нужен для выполнения задач описанных в GitHub Actions и GitLab CI. В GitHub нет необходимости подключать свои раннеры, потому что все на себя берут публичные раннеры. В GitLab время этих раннеров сильно ограничено, либо их может не быть совсем, если это GitLab в вашей инфраструктуре. При этом у вас нет публичных раннеров на компанию.

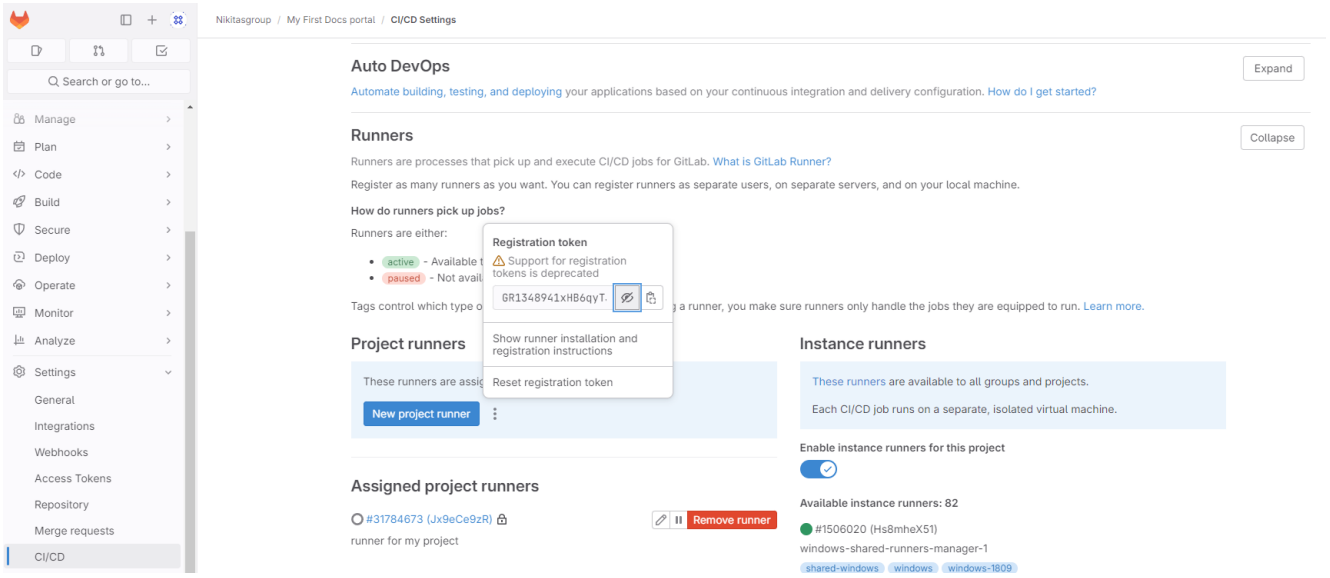
1. Нужно обязательно установить Docker — это самый простой способ поддерживать локальный раннер.
2. Все команды выполняются в командной строке. Сначала создадим пустую конфигурацию для нашего раннера.

   ```
   docker volume create gitlab-runner-config
   ```

3. Запустим контейнер с нашим раннером с помощью следующей команды:

   ```
   docker run -d --name gitlab-runner --restart always -v /var/run/docker.sock:/var/run/
   docker.sock -v gitlab-runner-config:/etc/gitlab-runner gitlab/gitlab-runner:alpine-v15.9.1
   ```

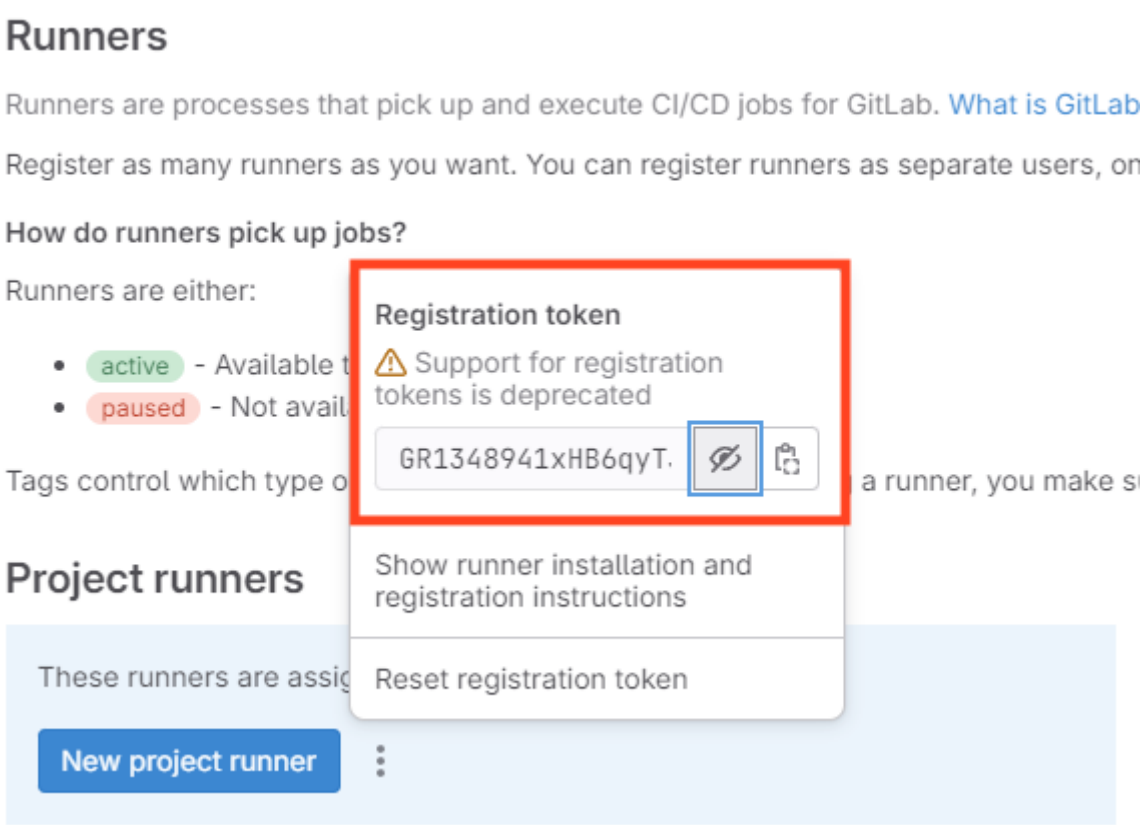
4. В репозитории проекта в **GitLab** перейдите в раздел **Settings → CI/CD** и разверните раздел **Runners**. Нам понадобится эта страница для настройки раннера:



5. Перейдите в Docker, в раздел Containers.
6. Найдите там контейнер gitlab-runner, кликните по нему и внутри перейдите на вкладку Exec.
7. Введите команду:

```
gitlab-runner register
```

8. Введите ваш GitLab instance, например: https://gitlab.com/
9. Введите registration token из шага 3.



10. Опционально оставляем описание и теги. Можно просто прокликать Enter.

11. Как executor пропишите: `docker`
12. Пропишите образ докера по умолчанию: `ruby 2.7`.
13. Обновите страницу в GitLab'е и проверьте, что раннер появился: