

Архитектура компьютеров и операционные системы | Операционные системы

Лабораторная работа № 3. Markdown

Мугари Абдеррахим - НКАбд-03-22

Содержание

1	Цель работы:	5
2	Задания:	6
3	Теоретическое введение:	7
3.1	Markdown:	7
3.2	Зачем это нужно?	7
4	Выполнение лабораторной работы:	8
4.1	Цель второй лабораторной работы:	8
4.2	Теоретическое введение:	8
4.2.1	Примеры использования git:	8
4.2.2	Выполнение второй лабораторной работы:	8
4.3	Контрольные вопросы:	16
4.4	выводы по результатам выполнения заданий:	20
5	Выводы, согласованные с целью работы:	21

Список иллюстраций

4.1	рисунок 1	9
4.2	рисунок 2	9
4.3	рисунок 3	10
4.4	рисунок 4	11
4.5	рисунок 5	11
4.6	рисунок 6	12
4.7	рисунок 7	13
4.8	рисунок 8	13
4.9	рисунок 9	13
4.10	рисунок 10	14
4.11	рисунок 11	14
4.12	рисунок 12	15
4.13	рисунок 13	15
4.14	рисунок 14	15
4.15	рисунок 15	17
4.16	рисунок 16	17
4.17	рисунок 17	18

Список таблиц

1 Цель работы:

- Научиться оформлять отчёты с помощью легковесного языка разметки **Markdown**.

2 Задания:

- Сделать отчёт по предыдущей лабораторной работе в формате **Markdown**.
- В качестве отчёта нужно предоставить отчёты в **3 форматах: pdf, docx и md** (в **архиве**, поскольку он должен содержать **скриншоты, Makefile** и т.д.)

3 Теоретическое введение:

3.1 Markdown:

- **Markdown** — язык разметки текстов. Такие тексты легко писать и читать. Их можно без труда сконвертировать в HTML. Большинство программистов предпочитают Markdown для написания документации, описаний своих проектов, написания блогов и так далее.

3.2 Зачем это нужно?

1. Для добавления разметки туда, где невозможна реальная разметка. Например, в простом текстовом файле или в тех же СМС, где невозможно выделение жирным, создание заголовков, выделение цитат и пр.
2. Для более удобного написания текстов для последующей конвертации в HTML или другие форматы.

4 Выполнение лабораторной работы:

4.1 Цель второй лабораторной работы:

- Изучить идеологию и применение средств контроля версий.
- Освоить умения по работе с git.

4.2 Теоретическое введение:

4.2.1 Примеры использования git:

- Система контроля версий **Git** представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды **git** с различными опциями.
- Благодаря тому, что **Git** является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

4.2.2 Выполнение второй лабораторной работы:

4.2.2.1 Установка программного обеспечения :

4.2.2.1.1 Установка git:

- На этом шаге мы должны были установить **git** через консоль (в нашем случае **git** уже был установлен) (рис. 4.1)


```
root@fedora:~  
[root@fedora ~]# dnf install git  
Last metadata expiration check: 1:59:55 ago on Sat 18 Feb 2023 07:23:51 PM MSK.  
Package git-2.39.2-1.fc37.x86_64 is already installed.  
Dependencies resolved.  
Nothing to do.  
Complete!  
[root@fedora ~]#
```

Рис. 4.1: рисунок 1

4.2.2.1.2 Установка gh:

- Затем нам пришлось скачать **gh** (рис. 4.2)

```
root@fedora:~  
[root@fedora ~]# dnf install gh  
Last metadata expiration check: 2:00:07 ago on Sat 18 Feb 2023 07:23:51 PM MSK.  
Dependencies resolved.  
=====
```

Package	Architecture	Version	Repository	Size
gh	x86_64	2.23.0-1.fc37	updates	8.3 M

```
=====
```

Installing:

Transaction Summary

=====

Install 1 Package

Total download size: 8.3 M
Installed size: 42 M
Is this ok [y/N]: y
Downloading Packages:
gh-2.23.0-1.fc37.x86 68% [=====] 1.2 MB/s | 5.7 MB 00:02 ETA

Рис. 4.2: рисунок 2

4.2.2.2 Базовая настройка git:

- на этом шаге я ничего не делал, потому что мой репозиторий уже был настроен. (рис. 4.3)

Базовая настройка git

- Зададим имя и email владельца репозитория:

```
git config --global user.name "Name Surname"
git config --global user.email "work@mail"
```

- Настроим utf-8 в выводе сообщений git:

```
git config --global core.quotePath false
```

- Настройте верификацию и подписание коммитов git (см. [Верификация коммитов git с помощью GPG](#)).
- Зададим имя начальной ветки (будем называть ее **master**):

```
git config --global init.defaultBranch master
```

- Параметр **autocrlf**:

```
git config --global core.autocrlf input
```

- Параметр **safecrlf**:

```
git config --global core.safecrlf warn
```

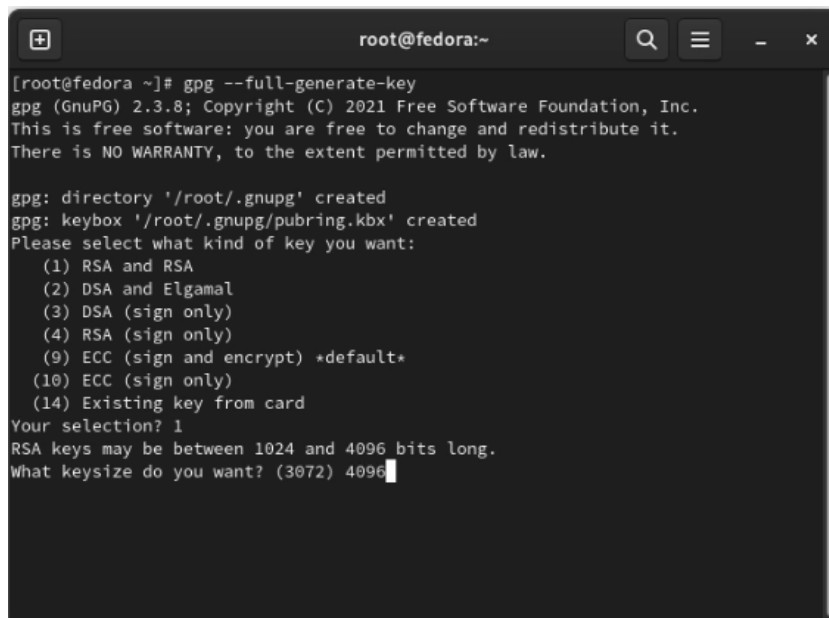
Рис. 4.3: рисунок 3

4.2.2.3 Создать ключи ssh:

- Для **ssh-ключа** было то же самое, поэтому мне пришлось пропустить этот шаг

4.2.2.4 Создать gpg ключ:

- На этом шаге мы должны были сгенерировать **** gpg ключ ****, введя следующие команды (рис. 4.4)



```
root@fedora:~  
[root@fedora ~]# gpg --full-generate-key  
gpg (GnuPG) 2.3.8; Copyright (C) 2021 Free Software Foundation, Inc.  
This is free software: you are free to change and redistribute it.  
There is NO WARRANTY, to the extent permitted by law.  
  
gpg: directory '/root/.gnupg' created  
gpg: keybox '/root/.gnupg/pubring.kbx' created  
Please select what kind of key you want:  
  (1) RSA and RSA  
  (2) DSA and Elgamal  
  (3) DSA (sign only)  
  (4) RSA (sign only)  
  (9) ECC (sign and encrypt) *default*  
  (10) ECC (sign only)  
  (14) Existing key from card  
Your selection? 1  
RSA keys may be between 1024 and 4096 bits long.  
What keysize do you want? (3072) 4096
```

Рис. 4.4: рисунок 4

- затем нам пришлось ввести кодовую фразу для защиты нашего ключа (рис. 4.5)

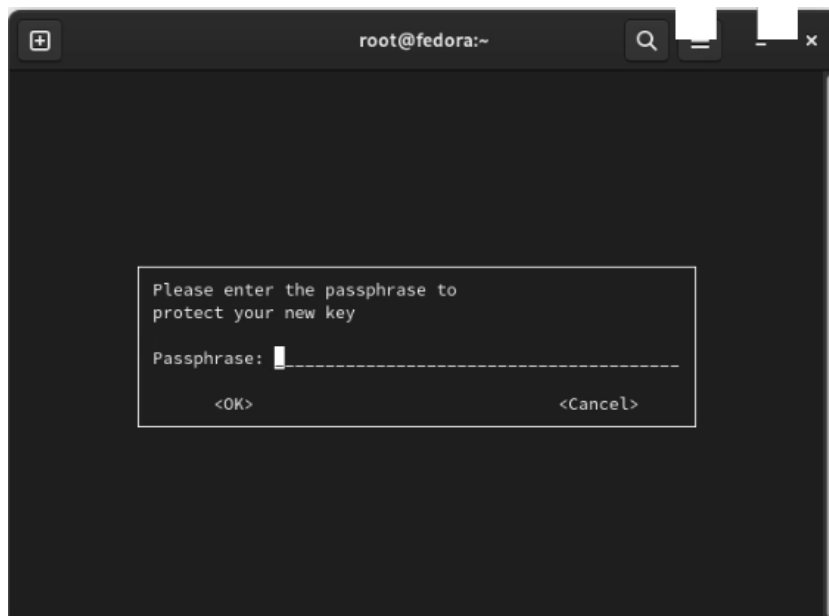


Рис. 4.5: рисунок 5

4.2.2.5 Настройка github:

- Я пропустил этот шаг, потому что я уже создал **учетную запись github** раньше.
- все мои данные были заполнены в учетной записи.(рис. 4.6)

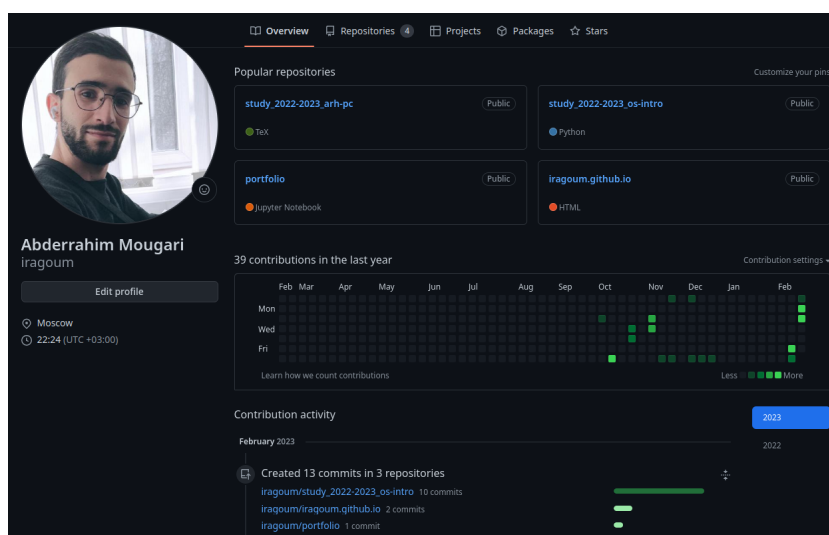
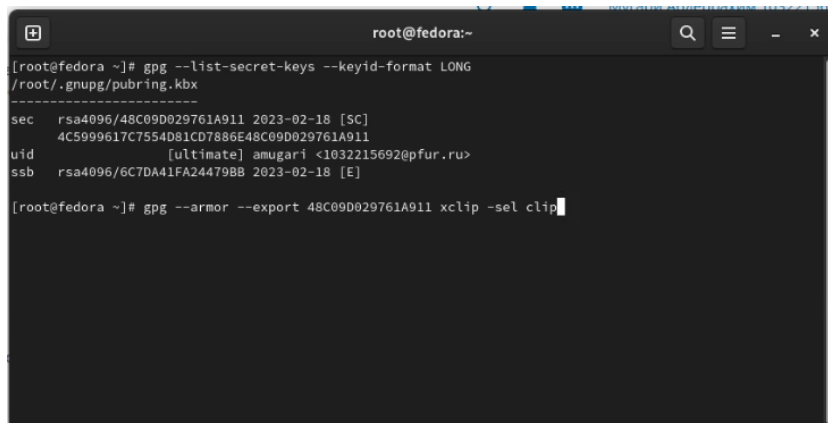


Рис. 4.6: рисунок 6

4.2.2.6 Добавление PGP ключа в GitHub:

- На этом шаге мы хотели добавить **ключ pgp** в нашу учетную запись github, поэтому нам пришлось скопировать отпечаток ключа (рис. 4.7)



```
root@fedora:~  
[root@fedora ~]# gpg --list-secret-keys --keyid-format LONG  
/root/.gnupg/pubring.kbx  
-----  
sec  rsa4096/48C09D029761A911 2023-02-18 [SC]  
      4C5999617C7554D81CD7886E48C09D029761A911  
uid          [ultimate] amugari <1032215692@pfur.ru>  
ssb  rsa4096/6C7DA41FA24479BB 2023-02-18 [E]  
  
[root@fedora ~]# gpg --armor --export 48C09D029761A911 xclip -sel clip
```

Рис. 4.7: рисунок 7

- после этого мы открыли **github** и в **настройках** добавили наш ключ **PGP** (рис. 4.8)

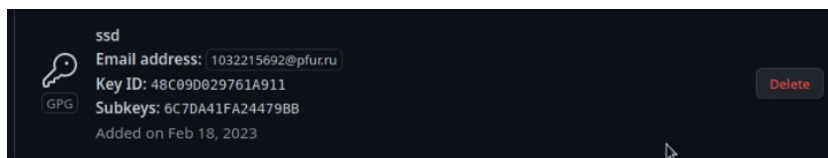


Рис. 4.8: рисунок 8

4.2.2.7 Настройка автоматических подписей коммитов git и gh:

- На этом шаге мы хотели заставить **github** использовать **наш введённый email**, в качестве подписи при отправке коммитов.
- для этого нам пришлось создать токен аутентификации, чтобы иметь доступ к нашей учетной записи git через консоль (рис. 4.9) (рис. 4.10)

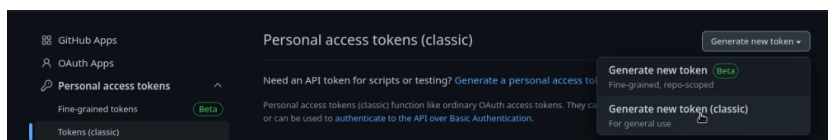


Рис. 4.9: рисунок 9

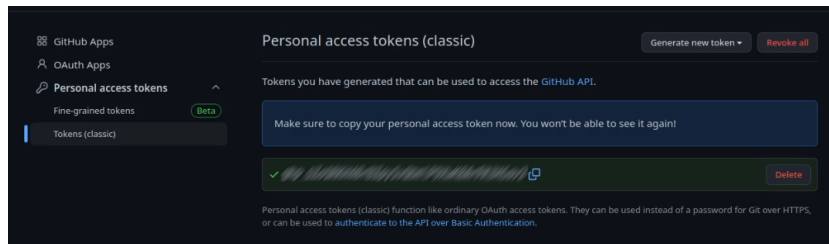


Рис. 4.10: рисунок 10

- и после этого мы вставили токен в консоль, которая дала нам доступ к нашей учетной записи github (рис. 4.11)

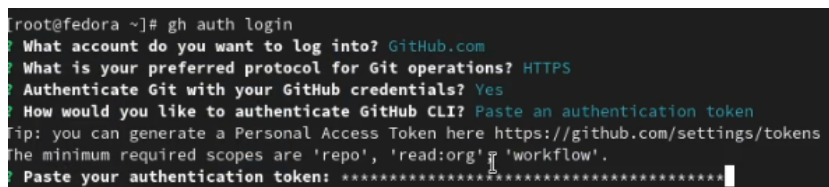


Рис. 4.11: рисунок 11

4.2.2.8 Создание репозитория курса на основе шаблона:

- Для меня этот путь `~/work/study/2022-2023/“Операционные системы”` уже был создан, поэтому я сразу ввел вторую команду.
- На этом шаге мне пришлось пройти аутентификацию с помощью **токена**, чтобы получить **доступ**, затем я клонировал глобальный репозиторий в свой локальный (рис. 4.12) (рис. 4.13)

```
amugari@fedora:~/work/study/2021-2022/Операционные системы
[amugari@fedora ~]$ cd work/
[amugari@fedora work]$ cd study/2021-2022/Операционные системы/
[amugari@fedora Операционные системы]$ gh repo create study_2022-2023_os-intro --template=yamadharma/course-directory-student-template --public
To get started with GitHub CLI, please run:  gh auth login
Alternatively, populate the GH_TOKEN environment variable with a GitHub API authentication token.
[amugari@fedora Операционные системы]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'workflow'.
? Paste your authentication token: *****
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as iragoum
[amugari@fedora Операционные системы]$
```

Рис. 4.12: рисунок 12

```
[amugari@fedora Операционные системы]$ git clone --recursive git@github.com:iragoum/study_2022-2023_os-intro.git os-intro
Cloning into 'os-intro'...
remote: Enumerating objects: 126, done.
remote: Counting objects: 100% (126/126), done.
remote: Compressing objects: 100% (110/110), done.
Receiving objects: 33% (42/126)
```

Рис. 4.13: рисунок 13

4.2.2.9 Настройка каталога курса:

- На этом шаге я попытался удалить файл package.json, но он уже был удален, поэтому я немедленно отправил другие изменения (рис. 4.14)

```
[amugari@fedora os-intro]$ echo os-intro > COURSE
make
make: Nothing to be done for 'all'.
[amugari@fedora os-intro]$ git add .
[amugari@fedora os-intro]$ git commit -am 'feat(main): make course structure'
[master 6213f55] feat(main): make course structure
1 file changed, 1 insertion(+)
[amugari@fedora os-intro]$ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 288 bytes | 288.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:iragoum/study_2022-2023_os-intro.git
a773d9f..6213f55 master -> master
[amugari@fedora os-intro]$
```

Рис. 4.14: рисунок 14

4.3 Контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?
 - Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
 - хранилище (repository, сокр. репо), или репозиторий, — место хранения всех версий и служебной информации.
 - commit : создание новой версии («сделать коммит», «закоммитить»)
 - история журнал : перечень версий можно вернуться к любой.
 - Рабочая копия (working copy или working tree) — текущее состояние файлов проекта, основанное на версии из хранилища (обычно на последней).
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.(рис. 4.15)

Виды систем контроля версий

Централизованные

- Простота использования.
- Вся история — всегда в едином общем хранилище.
- Нужно подключение к сети.
- Резервное копирование нужно только одному хранилищу.
- Удобство разделения прав доступа к хранилищу.
- Почти все изменения навсегда попадают в общее хранилище.

Распределенные

- Двухфазный commit:
 - 1) запись в локальную историю;
 - 2) пересылка изменений другим.
- Подключение к сети не нужно.
- Локальные хранилища могут служить резервными копиями.
- Локальное хранилище контролирует его владелец,
 - но общее — администратор.
- Возможна правка локальной истории перед отправкой на сервер.

Рис. 4.15: рисунок 15

4. Опишите действия с VCS при единоличной работе с хранилищем.(рис. 4.16)

Общее хранилище: централизованная VCS

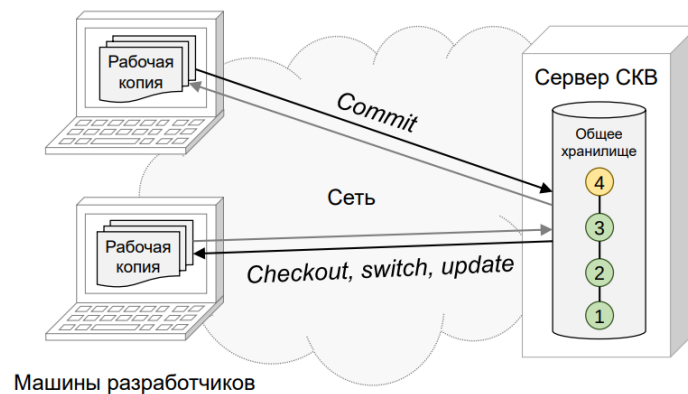


Рис. 4.16: рисунок 16

5. Опишите порядок работы с общим хранилищем VCS.(рис. 4.17)

Отдельные хранилища: распределенная VCS (DVCS)

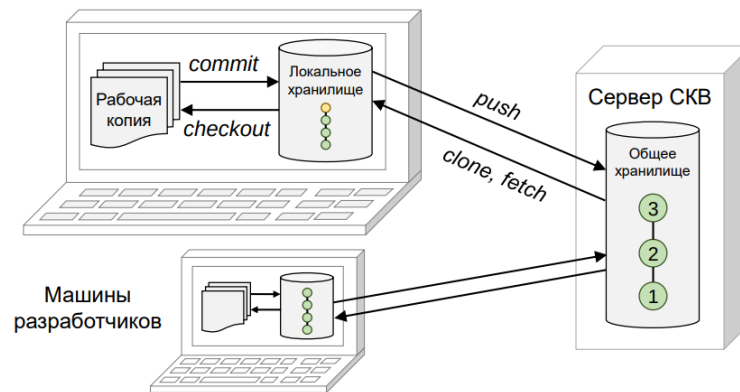


Рис. 4.17: рисунок 17

7. Назовите и дайте краткую характеристику командам git.

- Перечислим наиболее часто используемые команды git.
- Создание основного дерева репозитория:
- `git init`
- Получение обновлений (изменений) текущего дерева из центрального репозитория:
- `git pull`
- Отправка всех произведённых изменений локального дерева в центральный репозиторий:
- `git push`
- Просмотр списка изменённых файлов в текущей директории:
- `git status`

- Просмотр текущих изменений:
 - `git diff`
 - Сохранение текущих изменений:
 - добавить все изменённые и/или созданные файлы и/или каталоги:
 - `git add .`
 - добавить конкретные изменённые и/или созданные файлы и/или каталоги:
 - `git add имена_файлов`
 - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):
 - `git rm имена_файлов`
 - Сохранение добавленных изменений:
 - сохранить все добавленные изменения и все изменённые файлы:
 - `git commit -am 'Описание коммита'`
 - сохранить добавленные изменения с внесением комментария через встроенный редактор:
 - `git commit`
9. Что такое и зачем могут быть нужны ветви (branches)?
- Ветвь (branch) — это линейный участок истории.
 - Ветвь по умолчанию называется master. О том, зачем нужны другие ветви и как история может быть нелинейной, см. далее.
 - В выводе `git log` отмечен конец ветви.
10. Как и зачем можно игнорировать некоторые файлы при commit?

- Существуют различные типы файлов, которые мы, возможно, захотим, чтобы git проигнорировал перед фиксацией, например, файлы, связанные с нашими пользовательскими настройками или любыми настройками утилиты, личные файлы, такие как пароли и ключи API. Эти файлы никому больше не нужны, и мы не хотим загромождать наш git. Мы можем сделать это с помощью “.gitignore”

4.4 выводы по результатам выполнения заданий:

- после выполнения этих упражнений мы смогли применить на практике наши знания, которые мы получили о git и системе контроля версий в целом.

5 Выводы, согласованные с целью работы:

- к концу лабораторной работы этой лабораторной работе мы узнали, как использовать markdown для создания pdf-файлов быстрее и эффективнее.