Лабораторная работа №3

Markdown

Поляков Глеб Сергеевич

Содержание

1 Цель работы **5**

2 Задание **6**

3 Теоретическое введение **7**
3.1 Предварительные сведения 7
3.1.1 Базовые сведения о Markdown 7

4 This is heading 1 **8**
4.1 This is heading 2 8
4.1.1 This is heading 3 8
4.2 Обработка файлов в формате Markdown 10

5 Выполнение лабораторной работы **12**

6 Выводы **14**

Список литературы **15**

Список иллюстраций

5.1 Добавление цели работы 12
5.2 Добавление задания 12
5.3 Добавление теоретического введения 12
5.4 Добавление выполнения лабораторной работы 12
5.5 Добавление контрольных вопросов 13

Список таблиц

1 Цель работы

Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

2 Задание

- Сделайте отчёт по предыдущей лабораторной работе в формате Markdown.
- В качестве отчёта просьба 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 Предварительные сведения

3.1.1 Базовые сведения о Markdown

Чтобы создать заголовок, используйте знак (#), например:

4 This is heading 1

4.1 This is heading 2

4.1.1 This is heading 3

4.1.1.1 This is heading 4

Чтобы задать для текста полужирное начертание, заключите его в двойные звездочки:

This text is **bold**.

Чтобы задать для текста курсивное начертание, заключите его в одинарные звездочки:

This text is *italic*.

Чтобы задать для текста полужирное и курсивное начертание, заключите его в тройные звездочки:

This is text is both ***bold and italic***.

Блоки цитирования создаются с помощью символа >:> The drought had lasted now for ten million years, and the reign of ⊠ the terrible lizards had long since ended. Here on the Equator, in ⊠ the continent which would one day be known as Africa, the battle ⊠ for existence had reached a new climax of ferocity, and the victor ⊠ was not yet in sight. In this barren and desiccated land, only the ⊠ small or the swift or the fierce could flourish, or even hope to ⊠ survive.

Неупорядоченный (маркированный) список можно отформатировать с помощью звездочек или тире: - List item 1 - List item 2 - List item 3

Чтобы вложить один список в другой, добавьте отступ для элементов дочернего списка:

- List item 1
  - List item A
  - List item B
- List item 2

Упорядоченный список можно отформатировать с помощью соответствующих цифр:

1. First instruction
2. Second instruction
3. Third instruction

Чтобы вложить один список в другой, добавьте отступ для элементов дочернего списка:

1. First instruction
   1. Sub-instruction
   2. Sub-instruction
2. Second instruction

Синтаксис Markdown для встроенной ссылки состоит из части [link text], представляющей текст гиперссылки, и части (file-name.md) –URL-адреса или имени файла, на который дается ссылка: link text

Markdown поддерживает как встраивание фрагментов кода в предложение, так и их размещение между предложениями в виде отдельных огражденных блоков. Огражденные блоки кода — это простой способ выделить синтаксис для фрагментов кода. Общий формат огражденных блоков кода:

```
language your code goes in here
```

Верхние и нижние индексы:

записывается как

```
H~2~O
```

записывается как

```
2^10^
```

Внутритекстовые формулы делаются аналогично формулам LaTeX. Например, формула sin2(☒) + cos2(☒) = 1 запишется как $\sin^2(x) + \cos^2(x) = 1$ Выключные формулы: sin2(☒) + cos2(☒) = 1

{#eq:eq:sin2+cos2} со ссылкой в тексте «Смотри формулу (**??**).» записывается как

```
$$
\sin^2 (x) + \cos^2 (x) = 1
$$ {#eq:eq:sin2+cos2}
Смотри формулу ([-@eq:eq:sin2+cos2]).
```

4.2 Обработка файлов в формате Markdown

Для обработки файлов в формате Markdown будем использовать Pandoc https://pandoc.org/. Конкретно, нам понадобится программа pandoc-citeproc https://github.com/jgm/pandoc/releases, https://github.com/lierdakil/pandoc-crossref/releases.

Преобразовать файл README.md можно следующим образом: pandoc README.md -o README.pdfили так

```
pandoc README.md -o README.docxМожно использовать следующий Makefile:
FILES = $(patsubst %.md, %.docx, $(wildcard *.md)) FILES += $(patsubst %.md, %.pd
LATEX_FORMAT =
```

```
FILTER = --filter pandoc-crossref
%.docx: %.md
-pandoc "$<" $(FILTER) -o "$@"
%.pdf: %.md
-pandoc "$<" $(LATEX_FORMAT) $(FILTER) -o "$@"
all: $(FILES) @echo $(FILES)
clean:
-rm $(FILES) *~
```

5 Выполнение лабораторной работы

1. Добавление цели работы(рис. fig. 5.1).

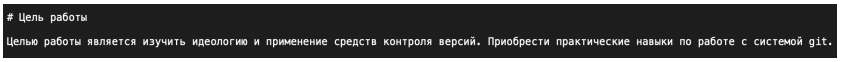


Рис. 5.1: Добавление цели работы

2. Добавление задания(рис. fig. 5.2).

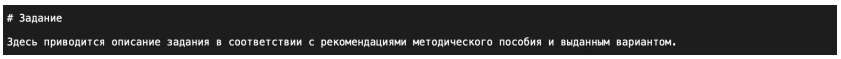


Рис. 5.2: Добавление задания

3. Добавление теоретического введения(рис. fig. 5.3).

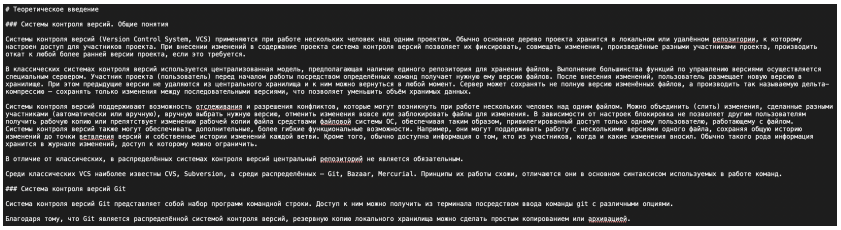


Рис. 5.3: Добавление теоретического введения

4. Добавление выполнения лабораторной работы(рис. fig. 5.4).

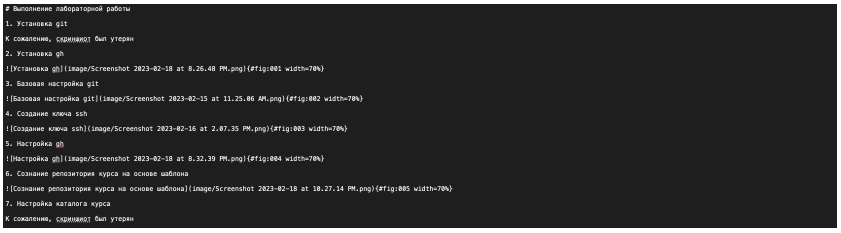
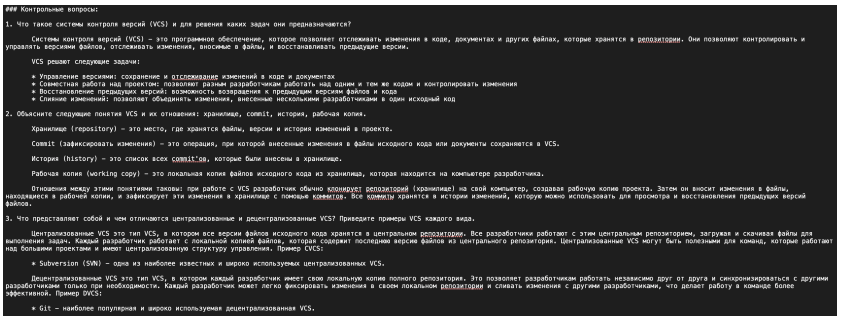


Рис. 5.4: Добавление выполнения лабораторной работы

5. Добавление контрольных вопросов(рис. fig. 5.5).



```
### Контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

      Системы контроля версий (VCS) — это программное обеспечение, которое позволяет отслеживать изменения в коде, документах и других файлах, которые хранятся в репозитории. Они позволяют контролировать и
управлять версиями файлов, отслеживать изменения, вносимые в файлы, и восстанавливать предыдущие версии.

      VCS решают следующие задачи:

      * Управление версиями: сохранение и отслеживание изменений в коде и документах
      * Совместная работа над проектом: позволяет разным разработчикам работать над одним и тем же кодом и контролировать изменения
      * Восстановление предыдущих версий: возможность возвращения к предыдущим версиям файлов и кода
      * Слияние изменений: позволяет объединять изменения, внесенные несколькими разработчиками в один исходный код

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

      Хранилище (repository) — это место, где хранятся файлы, версии и история изменений в проекте.

      Commit (зафиксировать изменения) — это операция, при которой внесенные изменения в файлы исходного кода или документы сохраняются в VCS.

      История (history) — это список всех commit'ов, которые были внесены в хранилище.

      Рабочая копия (working copy) — это локальная копия файлов исходного кода из хранилища, которая находится на компьютере разработчика.

      Отношения между этими понятиями таковы: при работе с VCS разработчик обычно клонирует репозиторий (хранилище) на свой компьютер, создавая рабочую копию проекта. Затем он вносит изменения в файлы,
находящиеся в рабочей копии, и зафиксирует эти изменения в хранилище с помощью комментов. Все комменты хранятся в истории изменений, которую можно использовать для просмотра и восстановления предыдущих версий
файлов.

3. Что представляет собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

      Централизованные VCS это тип VCS, в котором все версии файлов исходного кода хранятся в центральном репозитории. Все разработчики работают с этим центральным репозиторием, загружая и скачивая файлы для
выполнения задач. Каждый разработчик работает с локальной копией файлов, которая содержит последнюю версию файлов из центрального репозитория. Централизованные VCS могут быть полезными для команд, которые работают
над большими проектами и имеют централизованную структуру управления. Пример CVCS:

      * Subversion (SVN) — одна из наиболее известных и широко используемых централизованных VCS.

      Децентрализованные VCS это тип VCS, в котором каждый разработчик имеет свою локальную копию полного репозитория. Это позволяет разработчикам работать независимо друг от друга и синхронизироваться с другими
разработчиками только при необходимости. Каждый разработчик может легко фиксировать изменения в своем локальном репозитории и сливать изменения с другими разработчиками, что делает работу в команде более
эффективной. Пример DVCS:

      * Git — наиболее популярная и широко используемая децентрализованная VCS.
```

Рис. 5.5: Добавление контрольных вопросов

6 Выводы

Научился оформлять отчёты с помощью легковесного языка разметки Markdown.

Список литературы