

# **Шаблон отчёта по лабораторной работе №3**

**Работа с Markdown**

Гибшер Кирилл Владимирович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>5</b>
<b>2</b>	<b>Задание</b>	<b>6</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>7</b>
3.1	Базовые сведения о Markdown. . . . .	7
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>9</b>
<b>5</b>	<b>Выводы</b>	<b>13</b>
	<b>Список литературы</b>	<b>14</b>

## Список иллюстраций

4.1	Создание шапки отчета . . . . .	9
4.2	Цели и задачи . . . . .	9
4.3	Теоретическое введение 1 . . . . .	10
4.4	Теоретическое введение 2 . . . . .	11
4.5	Ход лабораторной работы . . . . .	11
4.6	Выводы . . . . .	12

## Список таблиц

# 1 Цель работы

- Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

## 2 Задание

1. Сделать отчет по предыдущей лабораторной работе в Markdown
2. Предоставить отчет в форматах doc,pdf,md.

## 3 Теоретическое введение

### 3.1 Базовые сведения о Markdown.

1. Чтобы создать заголовок, используйте знак ( # )
2. Чтобы задать для текста полужирное начертание, заключите его в двойные звездочки - This text is **bold**
3. Чтобы задать для текста курсивное начертание, заключите его в одинарные звездочки: *Cursiiiiiiiiiv check*
4. Чтобы задать для текста полужирное и курсивное начертание, заключите его в тройные звездочки: ***Vse svmeste***
5. Блоки цитирования создаются с помощью символа (>)

The drought had lasted now for ten million years

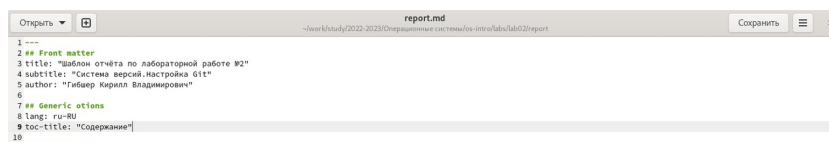
6. Неупорядоченный (маркированный) список можно отформатировать с помощью звездочек или тире
7. Упорядоченный список можно отформатировать с помощью соответствующих цифр
8. Чтобы вложить один список в другой, добавьте отступ для элементов дочернего списка

9. 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.
10. 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.
11. Нижние индексы записываются с помощью (~)  $H_2O$  , верхние же с помощью (^)  $2^{10}$
12. Внутритекстовые формулы делаются аналогично формулам LaTeX.



## 4 Выполнение лабораторной работы

Создаем шапку отчета. (рис. [4.1]).



```
1 ---
2 ## Front matter
3 title: "Шаблон отчёта по лабораторной работе №2"
4 subtitle: "Система версий. Настройка Git"
5 author: "Григорий Кирилл Владимирович"
6
7 ## Generic options
8 lang: ru-RU
9 toc-title: "Содержание"
10
11 ---
```

Рис. 4.1: Создание шапки отчета

Прописываем цели и задачи лаб работы. (рис. [4.2]).



```
100 ---
101 ## Цели работы
102
103 - Изучить идеологию и применение средств контроля версий.
104 - Освоить умения по работе с git.
105
106 ## Задание
107
108 1. Создать базовую конфигурацию для работы с git.
109 2. Создать ключ SSH.
110 3. Создать ключ PGP.
111 4. Настроить подписи git.
112 5. Зарегистрироваться на GitHub.
113 6. Создать локальный каталог для выполнения заданий по предмету.
114
```

Рис. 4.2: Цели и задачи

Пишем первую часть теоретического введения. (рис. [4.3]).

```

88
89 # Теоретическое введение
90
91 # Системы контроля версий. Общие понятия
92
93 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. В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальными сервером. Участники проекта (пользователи) перед началом работы посредством определённых команд получают нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельта-компрессию – сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединять (сливать) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменений. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или прерывает изменение рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, приватизированный доступ только одному пользователю, работающему с файлом. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Кроме того, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным. Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых – Git, Bazaar, Mercurial. Принципы их работы, отличаются они в основном синтаксисом используемых в работе команд.
94
95 # Основные команды git
96
97 Перечислим наиболее часто используемые команды git.
98
99 - Создание основного дерева репозитория: git init
100
101 - Получение обновлений (изменений) текущего дерева из центрального репозитория: git pull
102
103 - Отправка всех произведённых изменений локального дерева в центральный репозиторий: git push
104
105 - Просмотр списка изменённых файлов в текущей директории: git status
106
107 - Просмотр текущих изменений: git diff
108
109 - добавлять все изменённые и/или созданные файлы и/или каталоги: git add .
110
111 - добавлять конкретные изменённые и/или созданные файлы и/или каталоги: git add имена_файлов
112
113 - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена_файлов
114
115 - сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита'
116
117 - сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit
118
119 - создание новой ветки, базирующейся на текущей: git checkout -b имя_ветки
120
121 - переключение на некоторую ветку: git checkout имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой)
122
123 - отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки
124
125 - слияние ветки с текущим деревом: git merge --no-ff имя_ветки
126
127 - удаление локальной уже слитой с основным деревом ветки: git branch -d имя_ветки
128
129 - принудительное удаление локальной ветки: git branch -D имя_ветки
130
131 - удаление ветки с центрального репозитория: git push origin :имя_ветки
132
133 # Стандартные процедуры работы при наличии центрального репозитория
134
135 1. Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений): git checkout master | git pull | git checkout -b имя_ветки
136
137 2. Затем можно вносить изменения в локальное дерево и/или ветку.
138
139

```

Рис. 4.3: Теоретическое введение 1

Пишем вторую часть теоретического введения. (рис. [4.4]).

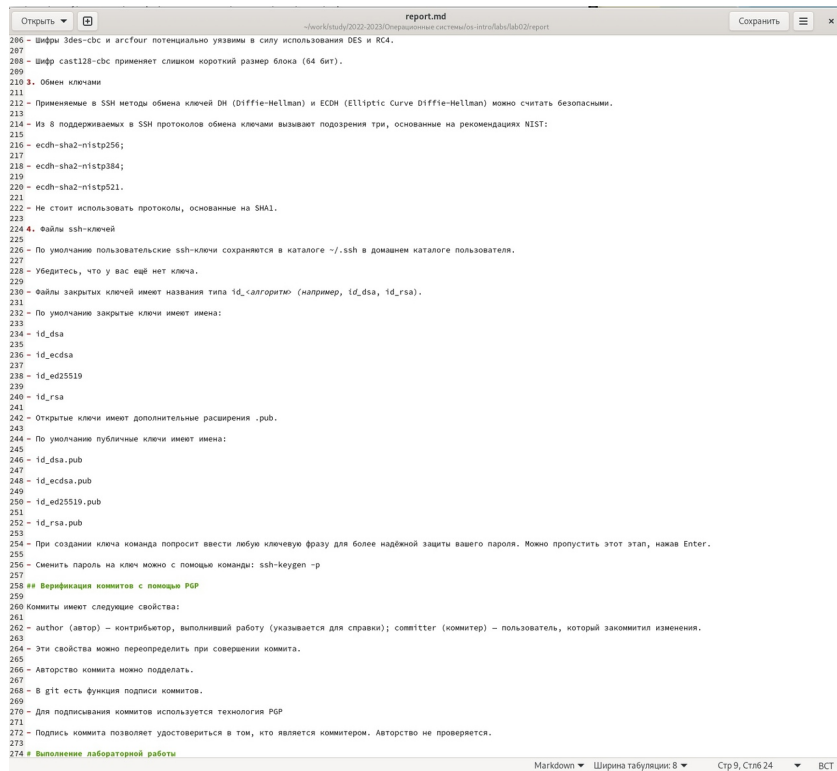


Рис. 4.4: Теоретическое введение 2

Прописываем ход лабораторной работы. рис. ([4.5]).

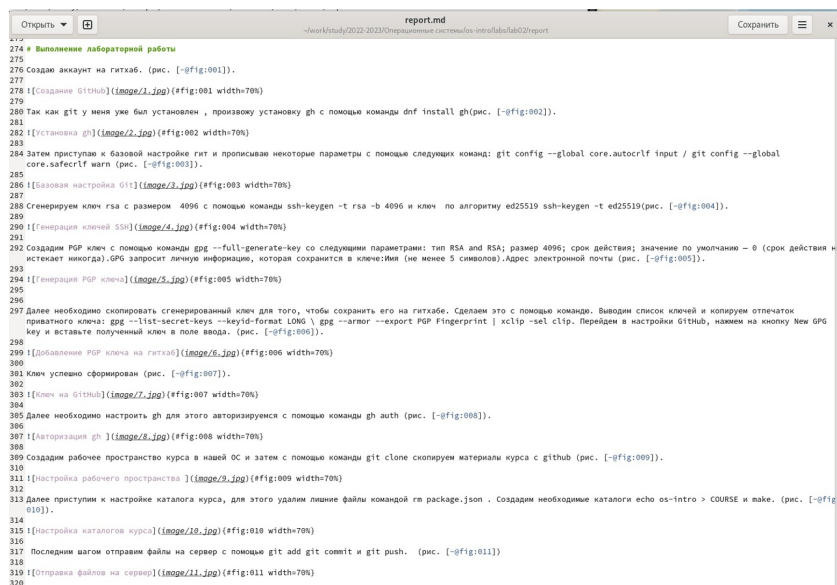


Рис. 4.5: Ход лабораторной работы

Пишем выводы и ссылки на научные источники. (рис. [4.6]).

```
389
390
391 # Выводы
392
393 Таким образом, благодаря данной лабораторной работе я приобрел практических навыков установки операционной системы на виртуальную машину, настройке минимально необходимых для
394 дальнейшей работы сервисов.
395
396 # Список литературы[unnumbered]
397
398 1. 0 системе контроля версий [Электронный ресурс] - Режим доступа:https://git-scm.com/book/ru/v2/Введение-0-системе-контроля-версий
399
400 2. Системы контроля версий [Электронный ресурс] - Режим доступа: http://u11.mpei.ru/study/courses/sdt/16/lecture02_2_vcs.slides.pdf.
401
402
403
404
405 ::: {#refs}
406 :::
Парная скобка найдена в строке: 396
```

Markdown Шрифт: табуляция: 8 Стр 396, Стб 33 ВСТ

Рис. 4.6: Выводы

## 5 Выводы

Я научился работать в языке Markdown , что пригодится мне для дальнейшей работы с документацией, а именно с отчетами по лабораторным работам!!!!

## Список литературы

1. Лабораторная работа №3 [Электронный ресурс] - Режим доступа:[https://esystem.rudn.ru/plab\\_markdown.pdf](https://esystem.rudn.ru/plab_markdown.pdf)