

# Лабораторная работа № 3. Markdown.

---

Гисматуллин Артём Вадимович

2023, 21 февраля

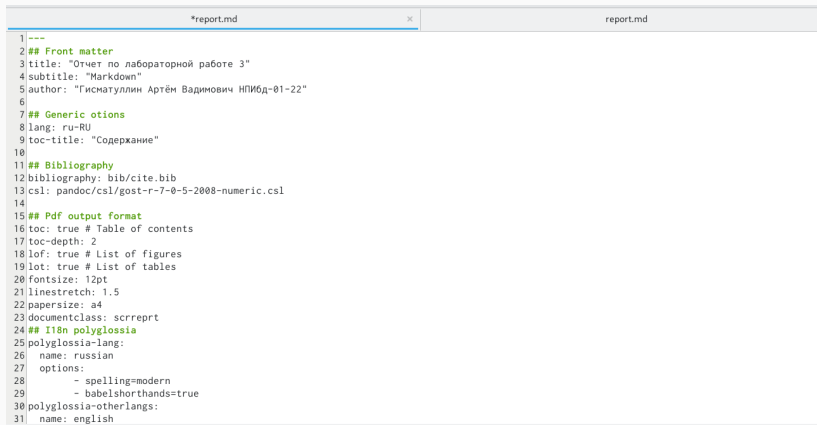
RUDN, Москва, Россия

- Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.
- Создать отчет по лабораторной работе №2.

## Ход работы

---

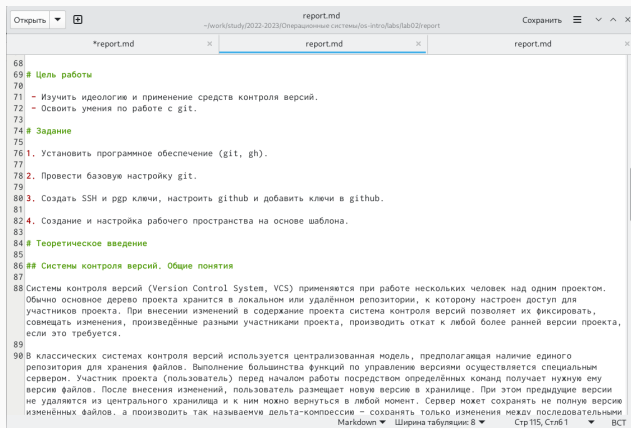
# Оформление титульного листа



The image shows a code editor with two tabs: '\*report.md' and 'report.md'. The code in the editor is as follows:

```
1 ---
2 ## Front matter
3 title: "Отчет по лабораторной работе 3"
4 subtitle: "Markdown"
5 author: "Гисматуллин Артём Вадимович НПИбд-01-22"
6
7 ## Generic options
8 lang: ru-RU
9 toc-title: "Содержание"
10
11 ## Bibliography
12 bibliography: bib/cite.bib
13 csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl
14
15 ## Pdf output format
16 toc: true # Table of contents
17 toc-depth: 2
18 lof: true # List of figures
19 lot: true # List of tables
20 fontsize: 12pt
21 linestretch: 1.5
22 papersize: a4
23 documentclass: scrreprt
24 ## I18n polyglossia
25 polyglossia-lang:
26   name: russian
27   options:
28     - spelling=modern
29     - babelshorthands=true
30 polyglossia-otherlangs:
31   name: english
```

Рис. 1: Markdown. Front matter

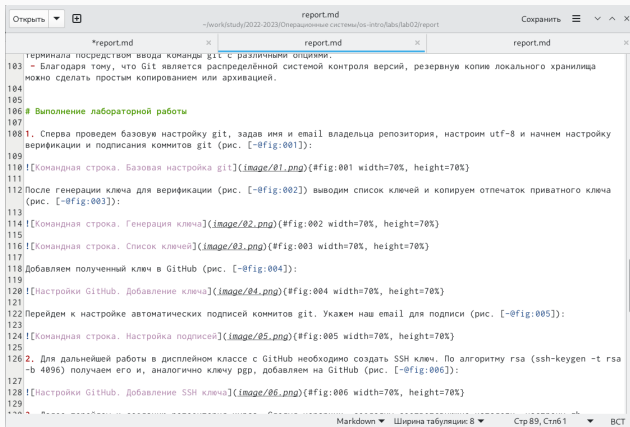


```
68
69 # Цель работы
70
71 - Изучить идеологию и применение средств контроля версий.
72 - Освоить умения по работе с git.
73
74 # Задание
75
76 1. Установить программное обеспечение (git, gh).
77
78 2. Провести базовую настройку git.
79
80 3. Создать SSH и pgr ключи, настроить github и добавить ключи в github.
81
82 4. Создание и настройка рабочего пространства на основе шаблона.
83
84 # Теоретическое введение
85
86 ## Системы контроля версий. Общие понятия
87
88 Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.
89 Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для
90 участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать,
91 совмещать изменения, произведённые разными участниками проекта, производить откат к любой более ранней версии проекта,
92 если это требуется.
93
94 В классических системах контроля версий используется централизованная модель, предполагающая наличие единого
95 репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным
96 сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает нужную ему
97 версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии
98 не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию
99 изменённых файлов, а производить так называемую дельта-компрессию – сохранять только изменения между последовательными
100 изменёнными файлами.
```

Markdown ▾ Ширина таблицы: 8 ▾ Стр 115, Стлб 1 ▾ ВСТ

Рис. 2: Mardown. Front matter

# Основная часть - выполнение работы



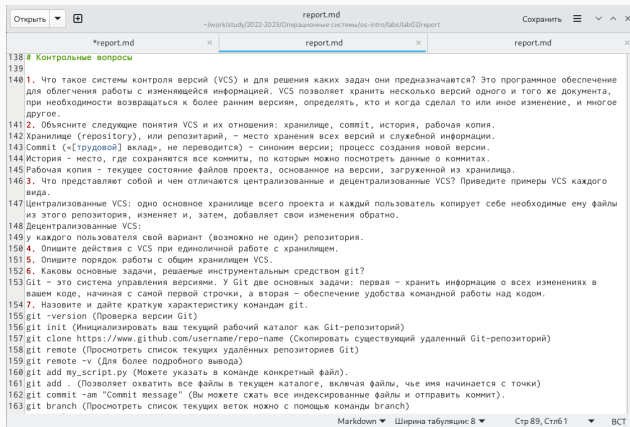
The screenshot shows a Markdown editor window titled "report.md" with the file path "~\work\study\2022-2023\Операционные системы\os-intro\labs\lab02\report". The editor contains a report in Russian about Git configuration. The text is as follows:

```
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.  
  
# Выполнение лабораторной работы  
  
1. Сначала проведем базовую настройку git, задав имя и email владельца репозитория, настроим utf-8 и начнем настройку  
верификации и подписания коммитов git (рис. [-@fig:001]):  
  
!Командная строка. Базовая настройка git (image/01.png) (#fig:001 width=70%, height=70%)  
  
12 После генерации ключа для верификации (рис. [-@fig:002]) выводим список ключей и копируем отпечаток приватного ключа  
(рис. [-@fig:003]):  
  
!Командная строка. Генерация ключа (image/02.png) (#fig:002 width=70%, height=70%)  
  
!Командная строка. Список ключей (image/03.png) (#fig:003 width=70%, height=70%)  
  
18 Добавляем полученный ключ в GitHub (рис. [-@fig:004]):  
  
!Настройки GitHub. Добавление ключа (image/04.png) (#fig:004 width=70%, height=70%)  
  
22 Перейдем к настройке автоматических подписей коммитов git. Укажем наш email для подписи (рис. [-@fig:005]):  
  
!Командная строка. Настройка подписей (image/05.png) (#fig:005 width=70%, height=70%)  
  
2. Для дальнейшей работы в дисплейном классе с GitHub необходимо создать SSH ключ. По алгоритму rsa (ssh-keygen -t rsa  
-b 4096) получаем его и, аналогично ключу pgr, добавляем на GitHub (рис. [-@fig:006]):  
  
!Настройки GitHub. Добавление SSH ключа (image/06.png) (#fig:006 width=70%, height=70%)
```

At the bottom of the editor, there is a status bar with the following information: "Markdown", "Ширина таблицы: 8", "Стр 89, Стлб 1", and "ВСТ".

Рис. 3: Markdown. Выполнение работы

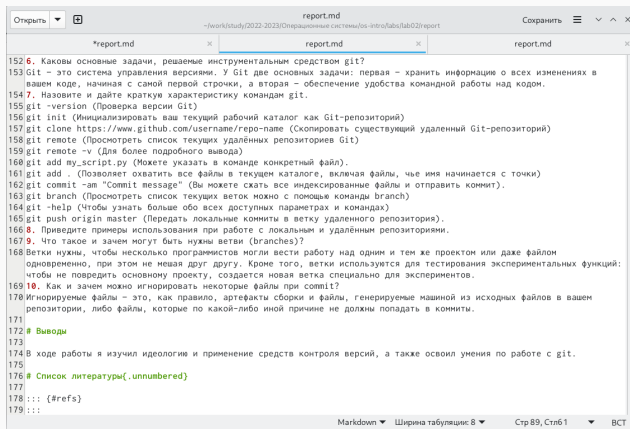
# Внимание на контрольные вопросы



```
138 # Контрольные вопросы
139
140 1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены? Это программное обеспечение для облегчения работы с изменяющейся информацией. VCS позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.
141 2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
142 Хранилище (repository), или репозиторий, – место хранения всех версий и служебной информации.
143 Commit («[трудо]клад», не переводится) – синоним версии; процесс создания новой версии.
144 История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах.
145 Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.
146 3. Что представляет собой и чем отличается централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.
147 Централизованные VCS: одно основное хранилище всего проекта и каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно.
148 Децентрализованные VCS:
149 у каждого пользователя свой вариант (возможно не один) репозитория.
150 4. Опишите действия с VCS при одиночной работе с хранилищем.
151 5. Опишите порядок работы с общим хранилищем VCS.
152 6. Каковы основные задачи, решаемые инструментальным средством git?
153 Git – это система управления версиями. У Git две основных задачи: первая – хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая – обеспечение удобства командной работы над кодом.
154 7. Назовите и дайте краткую характеристику командам git.
155 git -version (Проверка версии Git)
156 git init (Инициализировать ваш текущий рабочий каталог как Git-репозиторий)
157 git clone https://www.github.com/username/repo-name (Скопировать существующий удаленный Git-репозиторий)
158 git remote (Просмотреть список текущих удаленных репозиториях Git)
159 git remote -v (Для более подробного вывода)
160 git add my_script.py (Можете указать в команде конкретный файл).
161 git add . (Позволяет охватить все файлы в текущем каталоге, включая файлы, чье имя начинается с точки)
162 git commit -am "Commit message" (Вы можете скачать все индексированные файлы и отправить коммит).
163 git branch (Просмотреть список текущих веток можно с помощью команды branch)
```

Рис. 4: Markdown. Контрольные вопросы

# Выводы из выполнения работы



```
152 6. Каковы основные задачи, решаемые инструментальным средством git?
153 Git – это система управления версиями. У Git две основных задачи: первая – хранить информацию о всех изменениях в
вашем коде, начиная с самой первой строчки, а вторая – обеспечение удобства командной работы над кодом.
154 7. Назовите и дайте краткую характеристику командам git.
155 git -version (Проверка версии Git)
156 git init (Инициализировать ваш текущий рабочий каталог как Git-репозиторий)
157 git clone https://www.github.com/username/repo-name (Скопировать существующий удаленный Git-репозиторий)
158 git remote (Просмотреть список текущих удаленных репозиториях Git)
159 git remote -v (Для более подробного вывода)
160 git add my_script.py (Можете указать в команде конкретный файл).
161 git add . (Позволяет охватить все файлы в текущем каталоге, включая файлы, чье имя начинается с точки)
162 git commit -am "Commit message" (Вы можете скать все индексированные файлы и отправить коммит).
163 git branch (Просмотреть список текущих веток можно с помощью команды branch)
164 git -help (Чтобы узнать больше обо всех доступных параметрах и командах)
165 git push origin master (Передать локальные коммиты в ветку удаленного репозитория).
166 8. Приведите примеры использования при работе с локальным и удаленным репозиториями.
167 9. Что такое и зачем могут быть нужны ветви (branches)?
168 Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом
одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций:
чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.
169 10. Как и зачем можно игнорировать некоторые файлы при commit?
170 Игнорируемые файлы – это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем
репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты.
171
172 # Выводы
173
174 В ходе работы я изучил идеологию и применение средств контроля версий, а также освоил умения по работе с git.
175
176 # Список литературы{.unnumbered}
177
178 ::: {#refs}
179 :::
```

Рис. 5: Mardown. Выводы



# Формирование отчетов

```
avgismatullin@dk3n38 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab03/report $ make clean
rm report.docx report.pdf *~
rm: невозможно удалить 'report.pdf': Нет такого файла или каталога
rm: невозможно удалить '*~': Нет такого файла или каталога
make: [Makefile:34: clean] Ошибка 1 (игнорирование)

avgismatullin@dk3n38 ~/work/study/2022-2023/Операционные системы/os-intro/labs/lab03/report $ make
pandoc "report.md" --filter pandoc/filters/pandoc_fignos.py --filter pandoc/filters/pandoc_eqnos.py --filter pandoc/filters/pandoc_tablenos.py --filter pandoc/filters/pandoc_secnos.py --number-sections --citeproc -o "report.docx"

--main--: Bad reference: @fig:001.
--main--: Bad reference: @fig:002.
--main--: Bad reference: @fig:003.
--main--: Bad reference: @fig:004.

--main--: Bad reference: @fig:005.
pandoc "report.md" --filter pandoc/filters/pandoc_fignos.py --filter pandoc/filters/pandoc_eqnos.py --filter pandoc/filters/pandoc_tablenos.py --filter pandoc/filters/pandoc_secnos.py --pdf-engine=lualatex --pdf-engine-opt=--shell-escape --citeproc --number-sections -o "report.pdf"

--main--: Bad reference: @fig:001.
--main--: Bad reference: @fig:002.
--main--: Bad reference: @fig:003.
--main--: Bad reference: @fig:004.
--main--: Bad reference: @fig:005.
```

Рис. 6: Командная строка. Формирование отчетов

Успешно в ходе выполнения работы освоили процедуры оформления отчетов с помощью языка разметки Markdown.

Спасибо за понимание!