Отчет по лабораторной работе номер 2

Дидусь Кирилл Валерьевич

Содержание

# 1 Цель работы

Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

# 2 Задание

Лабораторная работа подразумевает использование markdown для создания отчетов для лабораторных работ.

# 3 Теория

Лабораторная работа является небольшой научно-исследовательской работой, которую и оформлять следует по всем утверждённым требованиям. При подготовке отчета по лабораторной работе вы освоите ряд важных элементов, которые в дальнейшем пригодятся вам при написании курсовой и дипломной работы.

Markdown (МФА: [ˈmɑːkdaʊn], произносится маркда́ун) — облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте, с максимальным сохранением его читаемости человеком, и пригодный для машинного преобразования в языки для продвинутых публикаций (HTML, Rich Text и других). Первоначально создан в 2004 году Джоном Грубером[en] и Аароном Шварцем.

# 4 Выполнение работы

1. Начнем подготовку отчета. Для экономии времени, воспользуемся шаблоном отчета из [репозитория](https://github.com/yamadharma/academic-laboratory-report-template/tree/master/report)

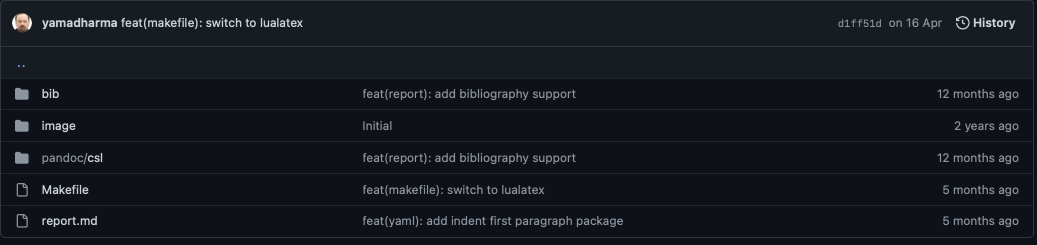


Рис. 1: рис.1. Репозиторий на GitHub.

1. Воспользуюсь программой **VScode** для редактирования отчета. Включу в отчет слудеющие синтаксические структуры:

* Список
* Нумерованный список
* Заголовки
* Ширный шрифт и курсив
* Ссылка
* Фрагмент кода
* Формула LaTeX
* Цитирование
* Вложенные изображения

1. Приведу фрагмент листинга программы для шифрования *методом Цезаря*.

def encrypt\_ceasar(input, shift):  
 output = ""  
 for char in input:  
 if char in eng\_aphabet:  
 output += eng\_aphabet[eng\_aphabet.index(char)+shift%len(eng\_aphabet)]  
 else: output += char  
 return output

1. Формула LaTeX:
2. Сохраним документ и сформируем отчет в форматах *.pdf* и *.docx*. Для этого выполним Makefile командой make в папке с отчетом.

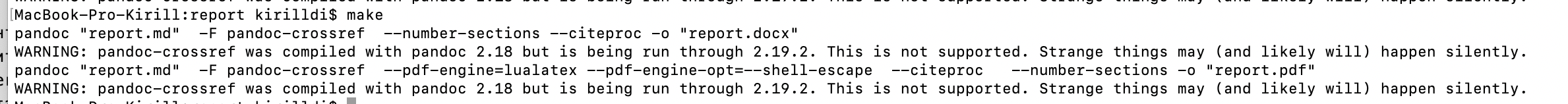


Рис. 2: рис.2. Формирование документов.

# 5 Библиография

1. ТУИС РУДН

# 6 Выводы

Во время выполнения лабораторной работы я освоил на практике формат markdown и сформировал отчет для лабораторной работы.

# 7 Ответы на контрольные вопросы для лабораторной номер 1

## 7.1 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.

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

• Хранилище (repository), или репозитарий, —место хранения всех версий и служебной информации. Commit — синоним версии; процесс создания новой версии. Рабочая копия (working copy) — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища (обычно на последней).

## 7.3 Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.

• Централизованные системы контроля версий представляют собой приложения типа клиент-сервер, когда репозиторий проекта существует в единственном экземпляре и хранится на сервере. Доступ к нему осуществлялся через специальное клиентское приложение. В качестве примеров таких программных продуктов можно привести CVS, Subversion. распределенные системы контроля версий (Distributed Version Control System, DVCS) позволяют хранить репозиторий (его копию) у каждого разработчика, работающего с данной системой. При этом можно выделить центральный репозиторий (условно), в который будут отправляться изменения из локальных и, с ним же эти локальные репозитории будут синхронизироваться. При работе с такой системой, пользователи периодически синхронизируют свои локальные репозитории с центральным и работают непосредственно со своей локальной копией. После внесения достаточного количества изменений в локальную копию они (изменения) отправляются на сервер. При этом сервер, чаще всего, выбирается условно, т. к. в большинстве DVCS нет такого понятия как “выделенный сервер с центральным репозиторием”. Еще пример - Wikipedia.

• В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. Пример — Bitcoin.

• В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

## 7.4 Опишите действия с VCS при единоличной работе с хранилищем.

• Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

git config –global user.name “Имя Фамилия”

git config –global user.email “work@mail”

и настроив utf-8 в выводе сообщений git:

git config –global quotepath false

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

cd

mkdir tutorial

cd tutorial

git init

## 7.5 Опишите порядок работы с общим хранилищем VCS.

• Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C “Имя Фамилия [work@mail](mailto:work@mail)”

Ключи сохраняться в каталоге~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

cat ~/.ssh/id\_rsa.pub | xclip -sel clip

вставляем ключ в появившееся на сайте поле.

## 7.6 Каковы основные задачи, решаемые инструментальным средством git?

• 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.

## 7.7 Назовите и дайте краткую характеристику командам git.

• 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 изменения:

git diff

– 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

– создание новой ветки, базирующейся на текущей: git checkout -b имя\_ветки

– переключение на некоторую ветку:

git checkout имя\_ветки

– отправка изменений конкретной ветки в центральный репозиторий:

git push origin имя\_ветки

– слияние ветки с текущим деревом:

git merge –no-ff имя\_ветки

## 7.8 Приведите примеры использования при работе с локальным и удалённым репозиториями.

• Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

git add hello.txt

git commit -am ‘Новый файл’

## 7.9 Что такое и зачем могут быть нужны ветви (branches)?

• Ветки нужны для того, чтобы программисты могли вести совместную работу над проектом и не мешать друг другу при этом. Кроме того, с помощью branches решаются следующие проблемы: нужно постоянно создавать архивы с рабочим кодом, сложно “переключаться” между архивами, сложно перетаскивать изменения между архивами, легко что-то напутать или потерять.

## 7.10 Как и зачем можно игнорировать некоторые файлы при commit?

• Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории. Во время работы над проектом эти файлы могут создаваться, но их не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл.gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов:

curl -L -s https://www.gitignore.io/api/list

Затем скачать шаблон, например, для C и C++:

curl -L -s https://www.gitignore.io/api/c >> .gitignore

curl -L -s https://www.gitignore.io/api/c++ >> .gitignore