Front matter

title: "Отчёт по лабораторной работе №3" subtitle: "Markdown" author: "Митичкина Екатерина Павловна"

Generic otions

lang: ru-RU toc-title: "Содержание"

Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt

I18n polyglossia

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

I18n babel

babel-lang: russian babel-otherlangs: english

Fonts

mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9

Biblatex

biblatex: true biblio-style: "gost-numeric" biblatexoptions: - parentracker=true - backend=biber - hyperref=auto - language=auto - autolang=other* - citestyle=gost-numeric

Pandoc-crossref LaTeX customization

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц" lolTitle: "Листинги"

Misc options

indent: true header-includes: - \usepackage{indentfirst} - \usepackage{float} # keep figures where there are in the text

- \floatplacement{figure}{H} # keep figures where there are in the text

Цель работы

Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

Задача

Сделайте отчёт по предыдущей лабораторной работе в формате Markdown.

Теоретическое введение:

Базовые сведения о Markdown Чтобы создать заголовок, используйте знак (#), например:

```
 # This is heading 1
## This is heading 2
### This is heading 3
#### 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,
```

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
1. Second instruction
1. Third instruction
```

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

```
1. First instruction
    1. Sub-instruction
    1. Sub-instruction
1. 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](file-name.md)`

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:

```
your code goes in here
```

Верхние и нижние индексы: H~2~O записывается как `H~2~O`

2^10^ записывается как `2^10^`

Внутритекстовые формулы делаются аналогично формулам LaTeX. Например, формула $\sin^2 (x) + \cos^2 (x) = 1$! запишется как `$\sin^2 (x) + \cos^2 (x) = 1$!`

Выключные формулы:

$\sin^2 (x) + \cos^2 (x) = 1$! {#eq:eq:sin2+cos2} со ссылкой в тексте «Смотри формулу ([-@eq:eq:sin2+cos2]).» записывается как

```
$$
\sin^2 (x) + \cos^2 (x) = 1
$$ {#eq:eq:sin2+cos2}

Смотри формулу ([-@eq:eq:sin2+cos2]).
```

Обработка файлов в формате Markdown

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

```
pandoc README.md -o README.pdf
```

или так

```
pandoc README.md -o README.docx
```

Можно использовать следующий Makefile

```
FILES = $(patsubst %.md, %.docx, $(wildcard *.md)) FILES += $(patsubst %.md, %.pdf, $(wildcard *.md))

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) *~
```

Выполнение лабораторной работы

Отчет по лабораторной работе №2

Цель работы

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

Задание

– Создать базовую конфигурацию для работы с git. – Создать ключ SSH. – Создать ключ PGP. – Настроить подписи git. – Зарегистрироваться на Github. – Создать локальный каталог для выполнения заданий попредмету.

Теоретическое введение

Системы контроля версий (Version Control System, VCS) применяются при работе несколь- ких 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 боль- шинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает 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 объединить (слить) изменения, сделанные разными участниками (авто- матически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользователям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом. Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например, они могут поддерживать работу с несколь- ми 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 распределён- ных — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

Выполнение лабораторной работы

Так как у меня уже была учетная запись на github, я пропустила пункт настройки github. Далее перешла к установке программного обеспечения. 1)

Установка git-flow в Fedora Linux Для этого я запустила код 2) Установка gh в Fedora Linux

Все программное обеспечение загрузилось Следящий пункт базовая настройка git Для начала я задала имя и email репозиторию: Настроила utf-8 в выводе сообщений git, задала имя начальной ветки, параметр autocrlf и safecrlf

После этого создала ключи ssh –по алгоритму rsa с ключём размером 4096 бит: –по алгоритму ed25519:

Следующий пункт был создать ключи pgp

Дальше нужно было добавить PGP ключа в GitHub Вывила список ключей и скопировала отпечаток приватного ключа

Скопировала сгенерированный PGP ключ в буфер обмена:

И вставила полученный код на github Дальше настроила автоматические подписи коммитов git

После нужно было авторизироваться Следующие создание репозитория курса на основе шаблона

И последнее это настройка каталога курса Перехожу в нужный каталог и удаляю не нужный файл, далее создаю нужный каталог и отправляю нужный файл на сервер

Выводы

В результате работы изучила идеологию и применение средств контроля версий. Освоила умения по работе с git.

Ответы на контрольные вопросы

1) Version Control System — программное обеспечение для облегчения работы с изменяющейся информацией. 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) В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определённых команд получает 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 неполную версию изменённых файлов, а производить так называемую дельта-компрессию—сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. 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.

3) Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. 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.

4) 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 сообщенийgit: 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

5) Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый): ssh-keygen -C"Имя Фамилия <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.

6) 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) Основные команды 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–просмотртекущих изменения: 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–создание новой ветки, базирующейся натекущей: git checkout -b имя*ветки–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) – отправка изменений конкретной ветки в центральный репозиторий: git push origin имя*ветки–слияние ветки стекущим деревом:git merge —no-ff имя*ветки–удаление ветки: – удаление локальной уже слитой с основным деревом ветки:git branch -d имя*ветки–принудительное удаление локальной ветки:git branch -D имя*ветки–удаление ветки с центрального репозитория: git push origin :имя_ветки

8) Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий): git add hello.txt git commit -am' Новый файл

9) Проблемы, которые решают ветки git: • нужно постоянно создавать архивы с рабочим кодом • сложно "переключаться" между архивами • сложно перетаскивать изменения между архивами • легко что-то напутать или потерять

10) Во время работы над проектом так или иначе могут создаваться файлы, которые 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 списки меняющихся 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

Вывод

В результате работы научилась оформлять отчёты с помощью легковесного языка разметки Markdown.