

Отчет по лабораторной работе №3

Markdown

Старовойтов Егор Сергеевич

Содержание

Цель работы	2
Задача	2
Теоретическое введение: Базовые сведения о Markdown	2
Выполнение лабораторной работы (создание отчета ко второй лабораторной работе) ..	4
Шаг 1 - Настройка служебной информации	4
Шаг 2 - Написание целей и задач	4
Шаг 3 - Подготовка скриншотов	5
Шаг 4 - Написание теоретического введение	5
Шаг 5 - Оформление повествования о ходе проделанной работы	6
Шаг 6 - Написание вывода	6
Шаг 7 - Ответы на вопросы	7
Отчет по лабораторной работе №2	7
Цель работы	7
Задание	7
Теоретическое введение	7
Системы контроля версий	7
Git	8
GitHub	8
Выполнение лабораторной работы	9
Шаг 1. Настройка github	9
Шаг 2. Установка программного обеспечения	9
Шаг 3. Базовая настройка git	11
Шаг 4. Создаем ключи ssh	11
Шаг 5. Создаем ключ PGP	12
Шаг 6. Добавляем PGP ключ в GitHub	13
Шаг 7. Настраиваем автоматическую подпись коммитов в git	14
Шаг 8. Настройка gh	14
Шаг 9	15

Вывод.....	15
Контрольные вопросы.....	15
1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены?.....	15
2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.....	16
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.	16
4. Опишите действия с VCS при единоличной работе с хранилищем.....	16
5. Опишите порядок работы с общим хранилищем VCS.	16
6. Каковы основные задачи, решаемые инструментальным средством git?	16
7. Назовите и дайте краткую характеристику командам git.	16
8. Приведите примеры использования при работе с локальным и удалённым репозиториями.	17
9. Что такое и зачем могут быть нужны ветви (branches)?	17
10. Как и зачем можно игнорировать некоторые файлы при commit?	17
Вывод.....	17

Цель работы

Научиться оформлять отчёты с помощью легковесного языка разметки Markdown.

Задача

Сделать отчет по предыдущей лабораторной работе в формате Markdown.

Теоретическое введение: Базовые сведения о Markdown

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

```
1 # This is heading 1
2 ## This is heading 2
3 ### This is heading 3
4 #### 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: 1 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:
Лабораторная работа № 3. Markdown

- 1 - List item 1
- 2 - List item A
- 3 - List item B
- 4 - 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)

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

```
1 FILES = $(patsubst %.md, %.docx, $(wildcard *.md))
2 FILES += $(patsubst %.md, %.pdf, $(wildcard *.md))
3
4 LATEX_FORMAT =
5
6 FILTER = --filter pandoc-crossref
7
8 %.docx: %.md
```

```

9 -pandoc "$<" $(FILTER) -o "$@"
10
11 %.pdf: %.md
12 -pandoc "$<" $(LATEX_FORMAT) $(FILTER) -o "$@"
13
14 all: $(FILES)
15 @echo $(FILES)
16
17 clean:
18 -rm $(FILES)

```

Выполнение лабораторной работы (создание отчета ко второй лабораторной работе)

Шаг 1 - Настройка служебной информации

Код ниже настраивает параметры pdf файла, который в дальнейшем будет получен с помощью программы pandoc:

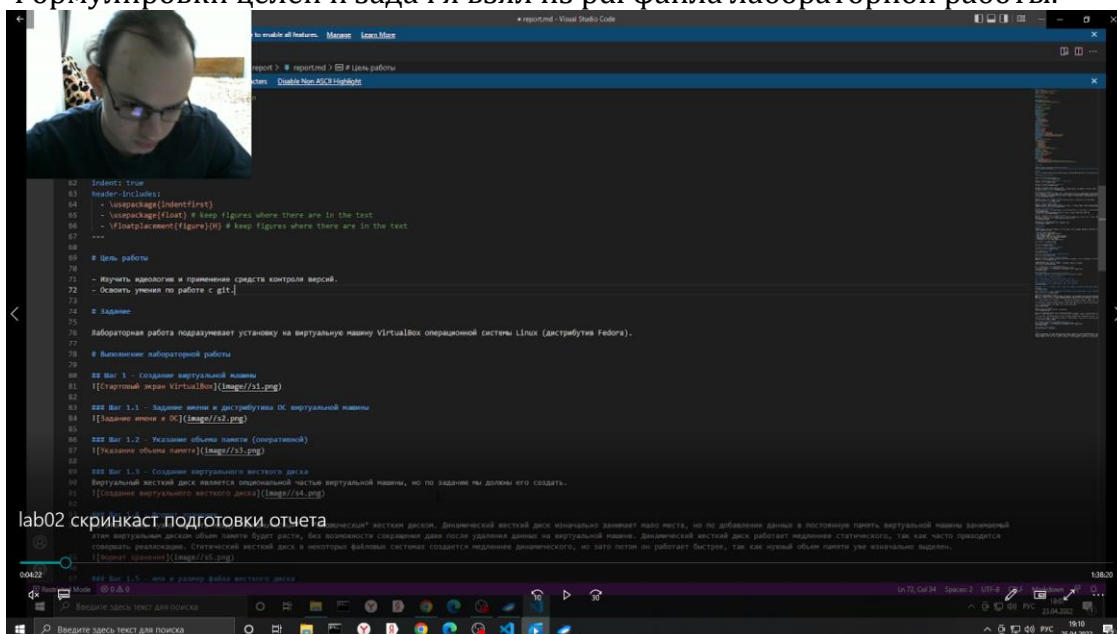
```

## Front matter
title: "Отчет по лабораторной работе №2"
subtitle: "Управление версиями"
author: "Старовойтов Егор Сергеевич"

```

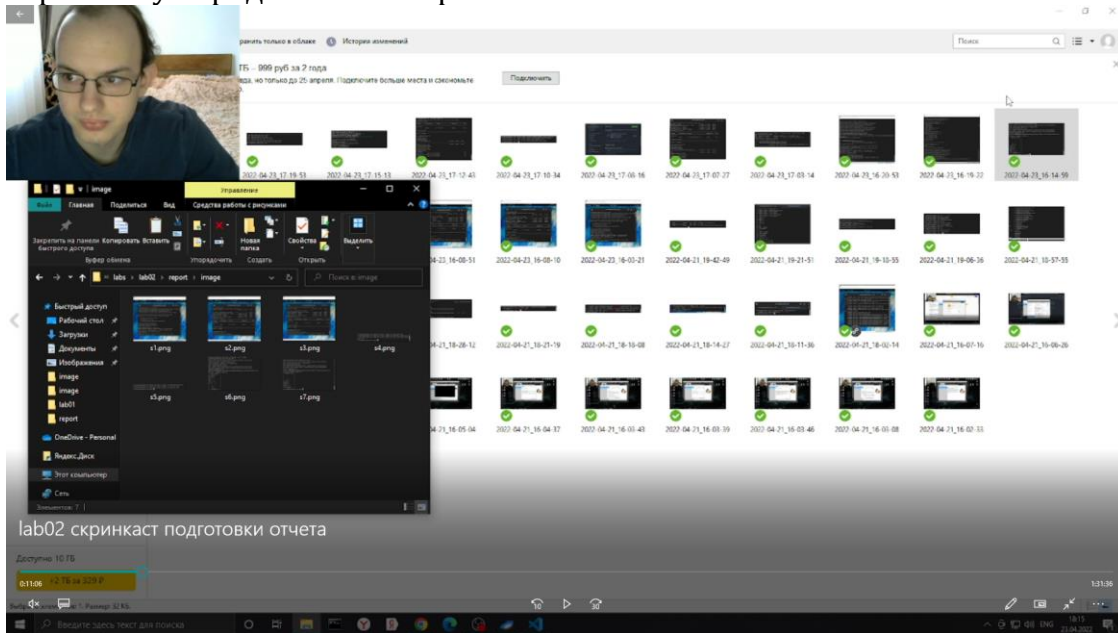
Шаг 2 - Написание целей и задач

Формулировки целей и задач я взял из pdf файла лабораторной работы.



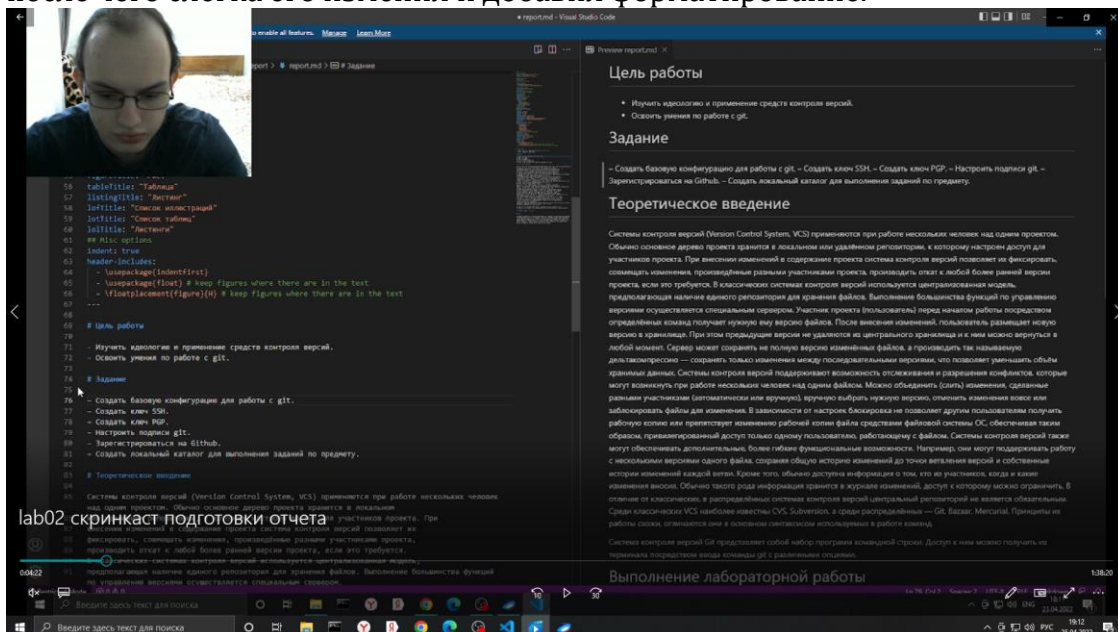
Шаг 3 - Подготовка скриншотов

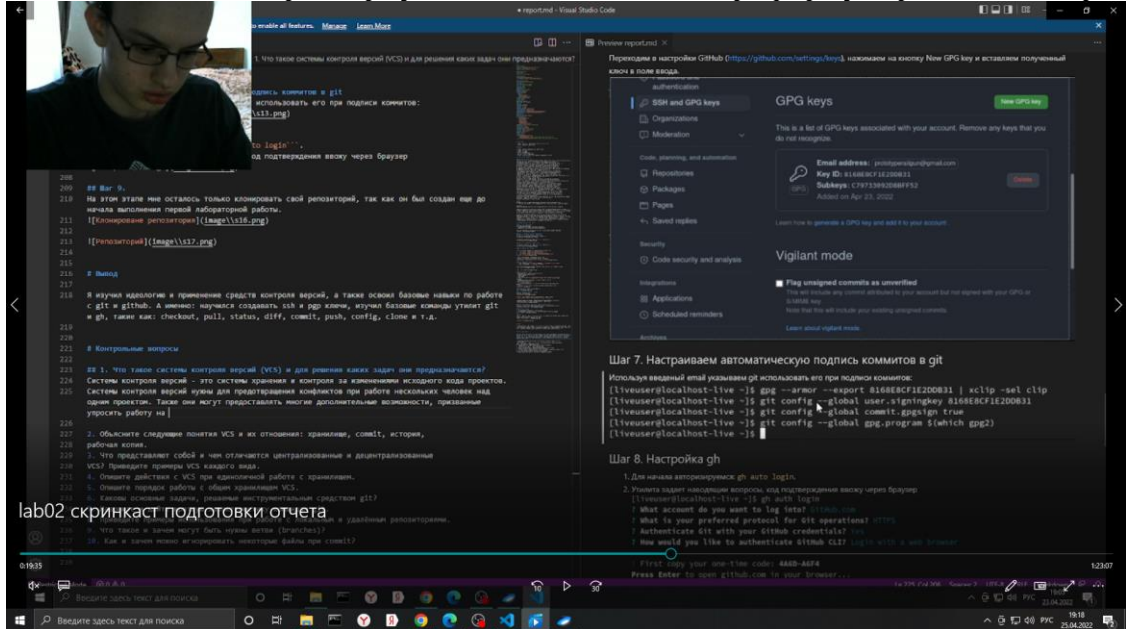
Я перенес все скриншоты в каталог /image рядом с отчетом report.md, дал каждому скриншоту порядковый номер.



Шаг 4 - Написание теоретического введение

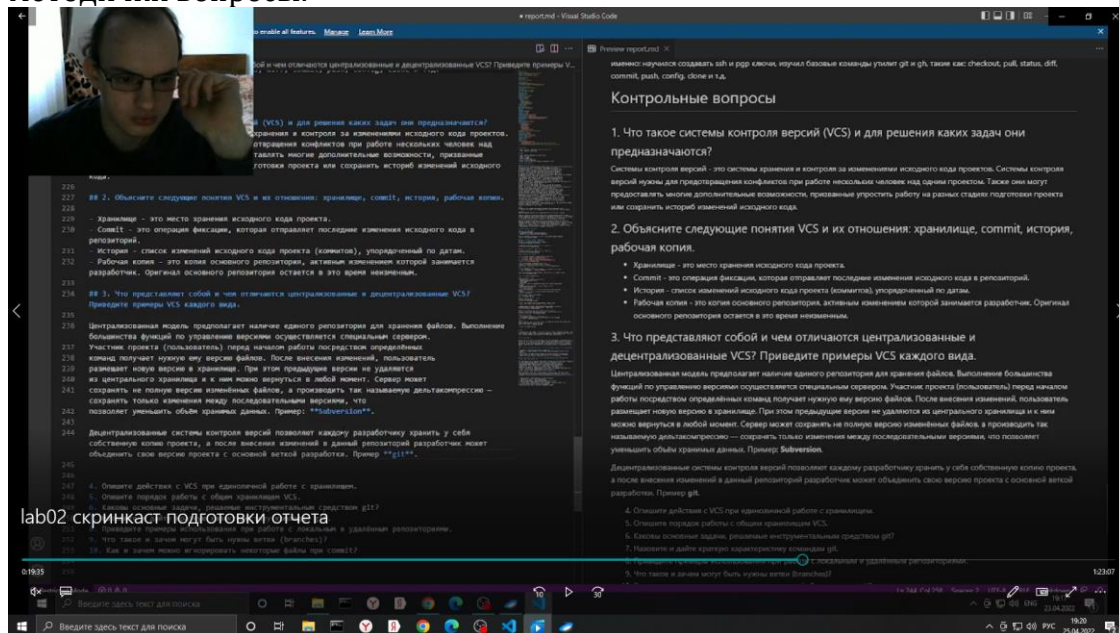
Текст теоретического введения я взял из pdf файла о второй лабораторной работе, после чего слегка его изменил и добавил форматирование.



[illegible]

Шаг 7 - Ответы на вопросы

После описания хода проделанной работы я ответил на все предложенные в тексте методички вопросы.



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 по предмету.

Теоретическое введение

Системы контроля версий

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 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 дельтакомпрессию — 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 (автоматически 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 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 схожи, отличаются они в основном синтаксисом используемых в работе команд.

Git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды `git` с различными опциями.

GitHub

GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Слоган сервиса — «Social Coding» — на русский можно перевести как «Пишем код вместе». На футболках же печатают совсем другую фразу: «Fork you!» («Ветвить тебя!»). С одной стороны, она созвучна с англоязычным ругательством и намекает на неформальную атмосферу. С другой, эти слова напоминают, что создавать новые форки

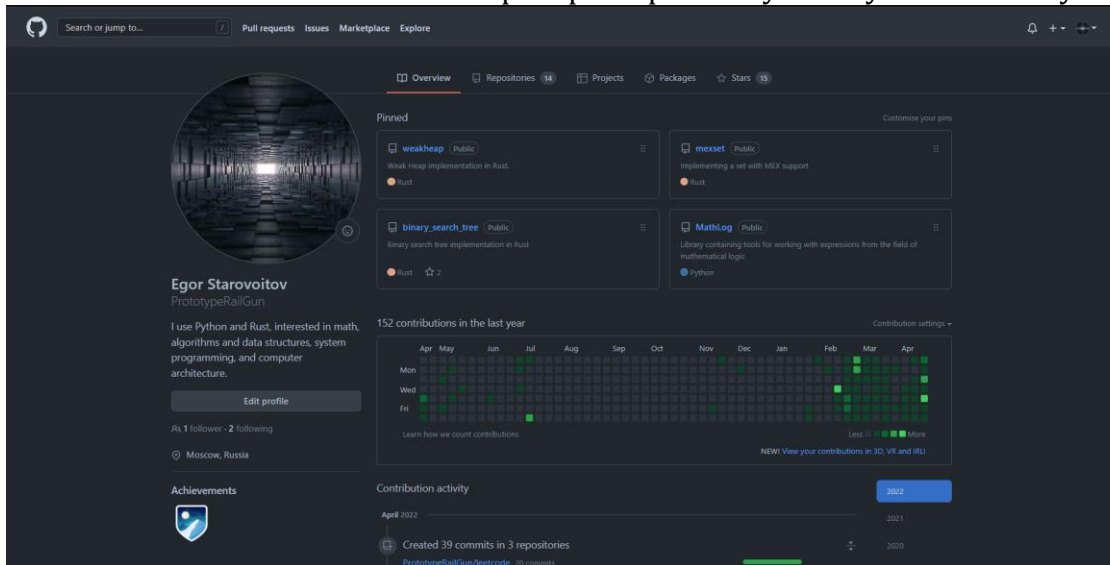
с Git можно легко и безболезненно — традиционно, к созданию веток разработки проектов с открытым исходным кодом относятся негативно — а также созвучна названию одной из возможностей GitHub — очереди форков.

Выполнение лабораторной работы

Шаг 1. Настройка github

Инструкция была следующей: 1. Создайте учётную запись на <https://github.com>. 2. Заполните основные данные на <https://github.com>.

На момент выполнения этой лабораторной работы у меня уже был аккаунт на GitHub.

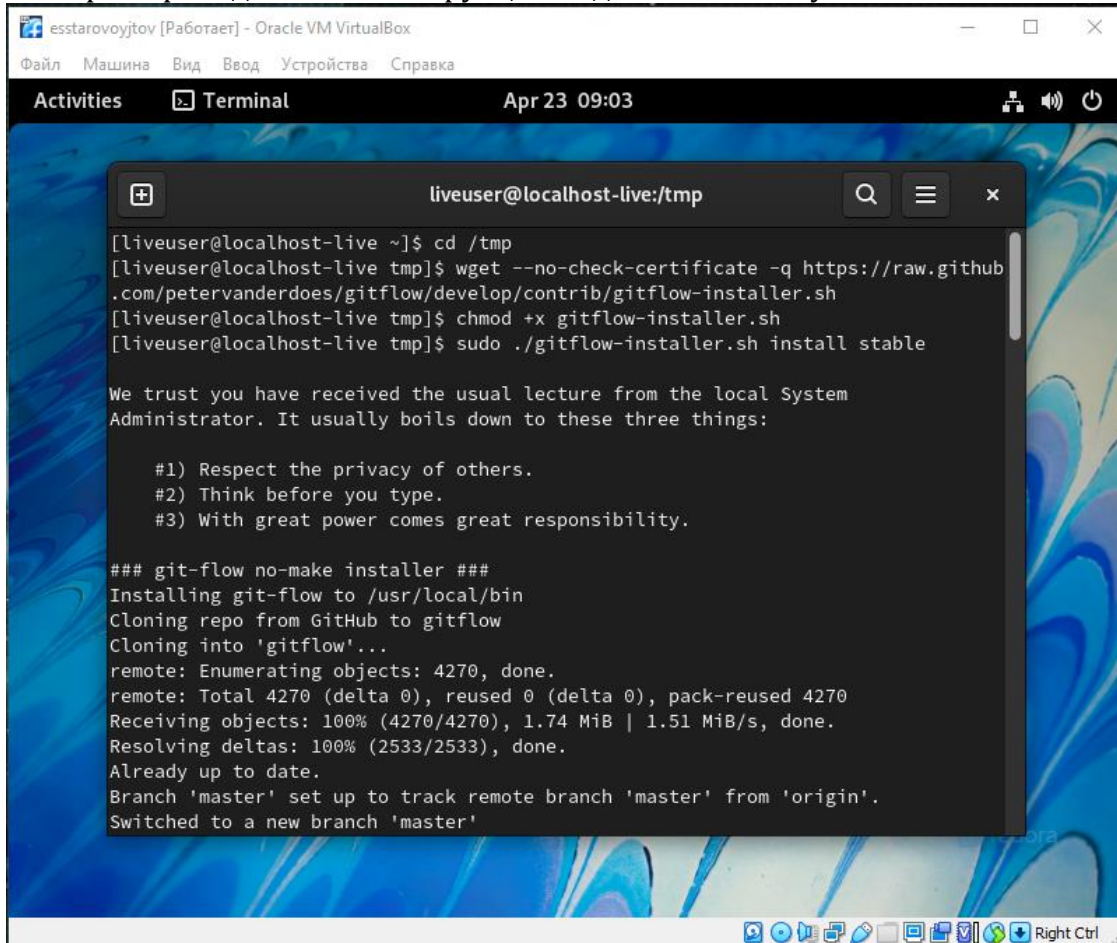


Шаг 2. Установка программного обеспечения

Шаг 2.1. Установка git-flow в Fedora Linux

– Это программное обеспечение удалено из репозитория. – Необходимо устанавливать его вручную.

Я набрал приведенный в инструкции код и вот что получилось:



The screenshot shows a terminal window titled "liveuser@localhost-live:/tmp" within an Oracle VM VirtualBox environment. The terminal displays the execution of several commands to install git-flow. The output includes a warning from the local System Administrator, a list of three principles (respect privacy, think before typing, and responsibility with power), and the successful cloning and installation of the git-flow repository to /usr/local/bin. The terminal also shows the setup of the 'master' branch to track the remote 'master' branch from 'origin' and switching to the new 'master' branch.

```
[liveuser@localhost-live ~]$ cd /tmp
[liveuser@localhost-live tmp]$ wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh
[liveuser@localhost-live tmp]$ chmod +x gitflow-installer.sh
[liveuser@localhost-live tmp]$ sudo ./gitflow-installer.sh install stable

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.

### git-flow no-make installer ###
Installing git-flow to /usr/local/bin
Cloning repo from GitHub to gitflow
Cloning into 'gitflow'...
remote: Enumerating objects: 4270, done.
remote: Total 4270 (delta 0), reused 0 (delta 0), pack-reused 4270
Receiving objects: 100% (4270/4270), 1.74 MiB | 1.51 MiB/s, done.
Resolving deltas: 100% (2533/2533), done.
Already up to date.
Branch 'master' set up to track remote branch 'master' from 'origin'.
Switched to a new branch 'master'
```

Шаг 2.2. Установка gh в Fedora Linux

Я набрал приведенный в инструкции код и вот что получилось:

```
[liveuser@localhost-live ~]$ sudo dnf install gh
Last metadata expiration check: 0:05:14 ago on Sat 23 Apr 2022 10:06:33 AM EDT.
Dependencies resolved.
=====
Package            Architecture      Version           Repository        Size
=====
Installing:
gh                 x86_64            2.7.0-1.fc35     updates           6.8 M

Transaction Summary
=====
Install 1 Package

Total download size: 6.8 M
Installed size: 32 M
Is this ok [y/N]: y
Downloading Packages:
gh-2.7.0-1.fc35.x86_64.rpm                6.9 MB/s | 6.8 MB    00:00
-----
Total                                       4.1 MB/s | 6.8 MB    00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing      :                                1/1
  Installing     : gh-2.7.0-1.fc35.x86_64         1/1
  Running scriptlet: gh-2.7.0-1.fc35.x86_64         1/1
  Verifying      : gh-2.7.0-1.fc35.x86_64         1/1

Installed:
gh-2.7.0-1.fc35.x86_64
```

Шаг 3. Базовая настройка git

- Настроим utf-8 и зададим имя и email владельца репозитория:

1. `git config --global core.quotepath false`
2. `git config --global user.name "Egor Starovoyjtov"`
3. `git config --global user.email "prototyperrailgun@gmail.com"`

```
[liveuser@localhost-live tmp]$ git config --global core.quotepath false
[liveuser@localhost-live tmp]$ git config --global user.name "Egor Starovoyjtov"
[liveuser@localhost-live tmp]$ git config --global user.email "prototyperrailgun@gmail.com"
[liveuser@localhost-live tmp]$
```

- Настройте верификацию и подписание коммитов git.
- Зададим имя начальной ветки (будем называть её master):

```
[liveuser@localhost-live tmp]$ git config --global init.defaultBranch master
[liveuser@localhost-live tmp]$ git config --global core.autocrlf input
[liveuser@localhost-live tmp]$ git config --global core.safecrlf warn
[liveuser@localhost-live tmp]$
```

Шаг 4. Создаем ключи ssh

- По алгоритму rsa с ключём размером 4096 бит: `ssh-keygen -t rsa -b 4096`
- По алгоритму ed25519: `ssh-keygen -t ed25519`

Я ввел пер

```
[liveuser@localhost-live tmp]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/liveuser/.ssh/id_rsa):
Created directory '/home/liveuser/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/liveuser/.ssh/id_rsa
Your public key has been saved in /home/liveuser/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:D0CF4AgNhF0NeIVlrum6Zml3ioDABjCe3KeWfYzAcw liveuser@localhost-live
The key's randomart image is:
+---[RSA 4096]-----+
|O=.o=B*oo|
|=E+*o=oo|
|.o*.o . .|
|oo = +|
|..B . o S|
|.o o . o|
|. . . . .|
| .=o. .|
| ++o.o|
+-----[SHA256]-----+
```

созданы.

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/liveuser/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/liveuser/.ssh/id_ed25519
Your public key has been saved in /home/liveuser/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:v76qWHqZW5DsFaiqXabOKXLQKXlyjCdClqly+4bxBG4 liveuser@localhost-live
The key's randomart image is:
+---[ED25519 256]---+
|
| .
| .
|.o .
|..oo.+ S
|=o++E..o
|+Ooooooooo.
|B=B=o+=.
|=BB=+.oo.o+.
+-----[SHA256]-----+
[liveuser@localhost-live tmp]$
```

Ключ ssh (ed25519)

Шаг 5. Создаем ключ PGP

- Генерируем ключ: `gpg --full-generate-key`
- Из предложенных опций выбираем:
- тип RSA and RSA;
- размер 4096;
- выбираем срок действия; значение по умолчанию— 0 (срок действия не истекает никогда).

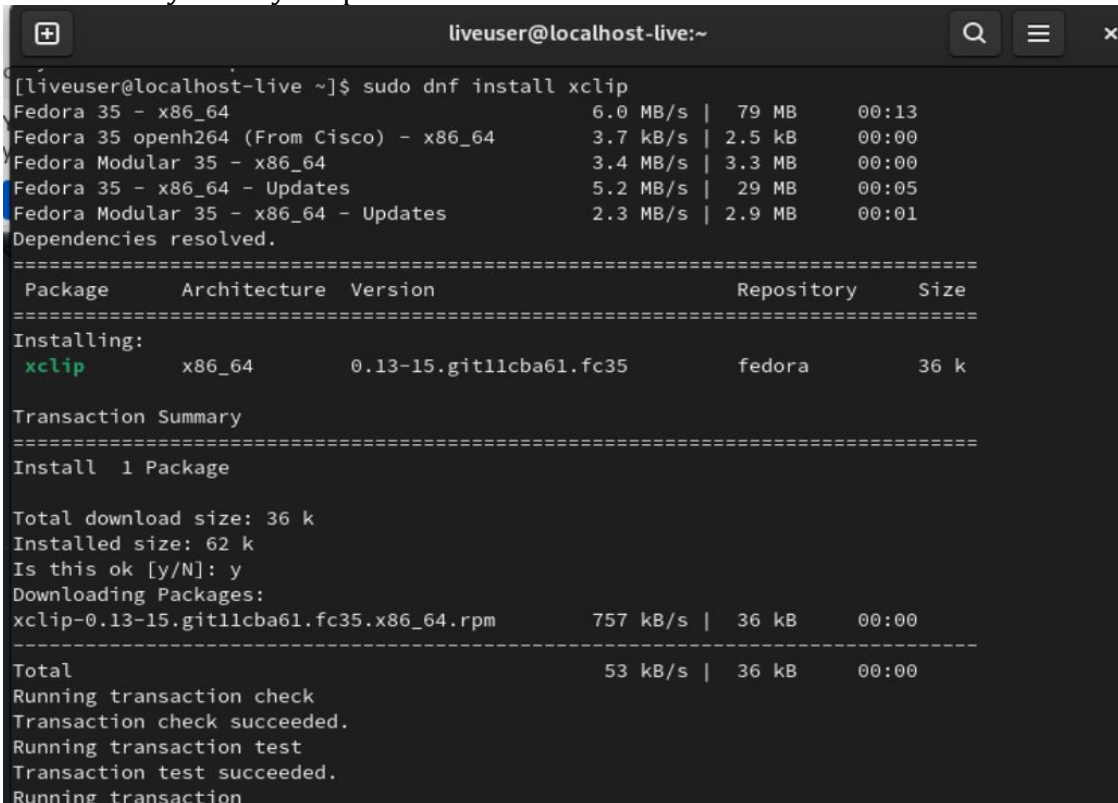
- GPG запросит личную информацию, которая сохранится в ключе:
- Имя (не менее 5 символов).
- Адрес электронной почты.
- При вводе email убедитесь, что он соответствует адресу, используемому на GitHub.
- Комментарий. Можно ввести что угодно или нажать клавишу ввода, чтобы оставить это поле пустым.

Я проследовал инструкции и создал PGP ключ.

```
[liveuser@localhost-live ~]$ gpg --list-secret-keys --keyid-format LONG
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
/home/liveuser/.gnupg/pubring.kbx
-----
sec  ed25519/8168E8CF1E2DDB31 2022-04-23 [SC]
      A7B8E7569EF18C4A3C5D98BA8168E8CF1E2DDB31
uid                          [ultimate] Egor Starovoyjtov <prototypemail@gmail.com>
ssb  cv25519/C79733092D8BFF52 2022-04-23 [E]
```

Шаг 6. Добавляем PGP ключ в GitHub

Скачиваем утилиту xclip:

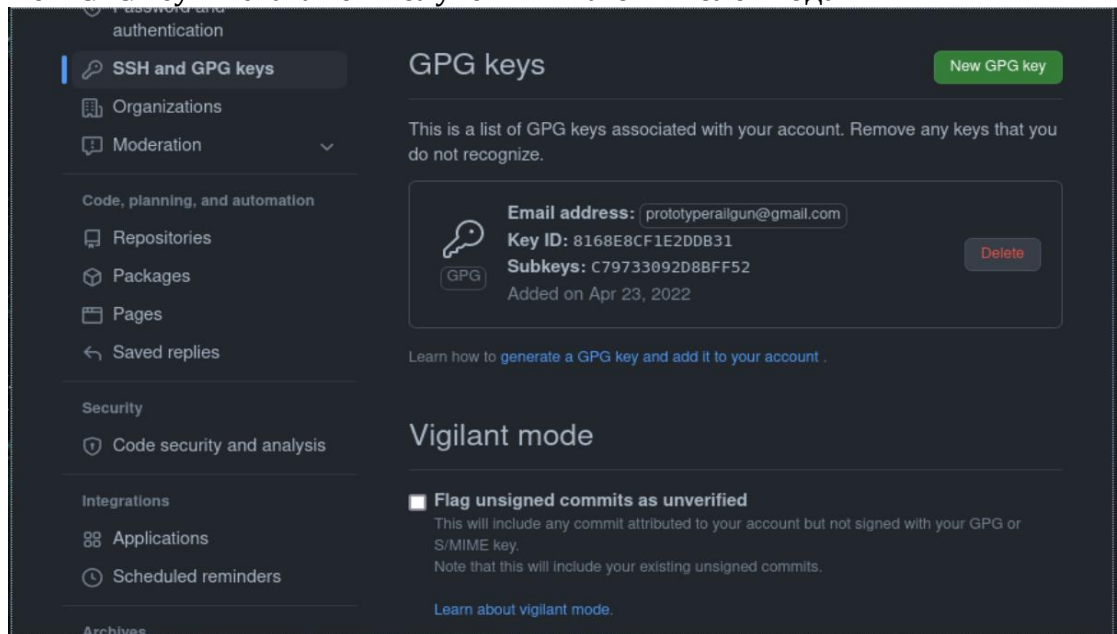


```
[liveuser@localhost-live ~]$ sudo dnf install xclip
Fedora 35 - x86_64                                6.0 MB/s | 79 MB      00:13
Fedora 35 openh264 (From Cisco) - x86_64         3.7 kB/s | 2.5 kB     00:00
Fedora Modular 35 - x86_64                       3.4 MB/s | 3.3 MB     00:00
Fedora 35 - x86_64 - Updates                      5.2 MB/s | 29 MB      00:05
Fedora Modular 35 - x86_64 - Updates              2.3 MB/s | 2.9 MB     00:01
Dependencies resolved.
=====
Package      Architecture Version                      Repository      Size
=====
Installing:
xclip        x86_64      0.13-15.git11cba61.fc35     fedora          36 k
Transaction Summary
=====
Install 1 Package

Total download size: 36 k
Installed size: 62 k
Is this ok [y/N]: y
Downloading Packages:
xclip-0.13-15.git11cba61.fc35.x86_64.rpm          757 kB/s | 36 kB      00:00
-----
Total                                              53 kB/s | 36 kB      00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
```

С помощью xclip копируем сгенерированный PGP ключ в буфер обмена: `gpg --armor --export <PGP Fingerprint> | xclip -sel clip`

Переходим в настройки GitHub (<https://github.com/settings/keys>), нажимаем на кнопку New GPG key и вставляем полученный ключ в поле ввода.



Шаг 7. Настраиваем автоматическую подпись коммитов в git

Используя введенный email указываем git использовать его при подписи коммитов:

```
[liveuser@localhost-live ~]$ gpg --armor --export 8168E8CF1E2DDB31 | xclip -sel clip
[liveuser@localhost-live ~]$ git config --global user.signingkey 8168E8CF1E2DDB31
[liveuser@localhost-live ~]$ git config --global commit.gpgsign true
[liveuser@localhost-live ~]$ git config --global gpg.program $(which gpg2)
[liveuser@localhost-live ~]$
```

Шаг 8. Настройка gh

1. Для начала авторизируемся: `gh auto login`.
2. Утилита задает наводящие вопросы, код подтверждения ввожу через браузер

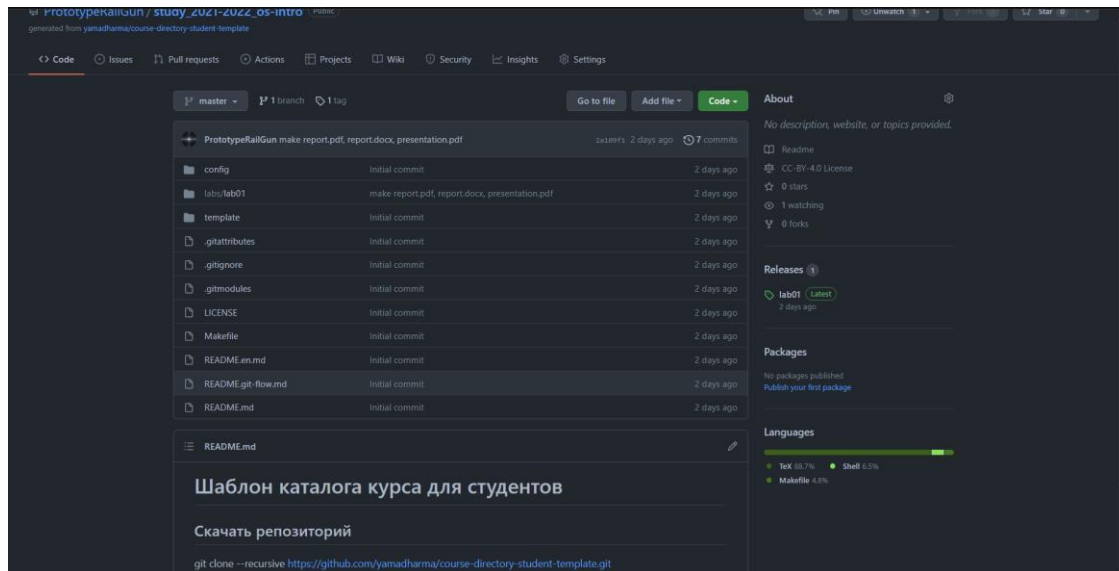
```
[liveuser@localhost-live ~]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 4A6D-A6F4
Press Enter to open github.com in your browser...
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as PrototyperrailGun
[liveuser@localhost-live ~]$
```


Шаг 9.

На этом этапе мне осталось только клонировать свой репозиторий, так как он был создан еще до начала выполнения первой лабораторной работы.

```
[liveuser@localhost-live ~]$ git clone https://github.com/PrototypeRailGun/study_2021-2022_os-intro
Cloning into 'study_2021-2022_os-intro'...
remote: Enumerating objects: 88, done.
remote: Counting objects: 100% (88/88), done.
remote: Compressing objects: 100% (71/71), done.
remote: Total 88 (delta 15), reused 75 (delta 7), pack-reused 0
Receiving objects: 100% (88/88), 35.71 MiB | 1.60 MiB/s, done.
Resolving deltas: 100% (15/15), done.
[liveuser@localhost-live ~]$
```



Репозиторий

Вывод

Я изучил идеологию и применение средств контроля версий, а также освоил базовые навыки по работе с git и github. А именно: научился создавать ssh и pgr ключи, изучил базовые команды утилит git и gh, такие как: checkout, pull, status, diff, commit, push, config, clone и т.д.

Контрольные вопросы

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

Системы контроля версий - это системы хранения и контроля за изменениями исходного кода проектов. Системы контроля версий нужны для предотвращения конфликтов при работе нескольких человек над одним проектом. Также они могут предоставлять многие дополнительные возможности, призванные упростить работу на разных стадиях подготовки проекта или сохранить историю изменений исходного кода.

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

- Хранилище - это место хранения исходного кода проекта.
- Commit - это операция фиксации, которая отправляет последние изменения исходного кода в репозиторий.
- История - список изменений исходного кода проекта (коммитов), упорядоченный по датам.
- Рабочая копия - это копия основного репозитория, активным изменением которой занимается разработчик. Оригинал основного репозитория остается в это время неизменным.

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

Централизованная модель предполагает наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером. Участник проекта (пользователь) перед началом работы посредством определенных 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 дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Пример: **Subversion**.

Децентрализованные системы контроля версий позволяют каждому разработчику хранить у себя собственную копию проекта, а после внесения изменений в данный репозиторий разработчик может объединить свою версию проекта с основной веткой разработки. Пример: **git**.

4. Опишите действия с VCS при единоличной работе с хранилищем.

1. Клонирование репозитория на свой компьютер.
2. Внесение изменений в код проекта.
3. Фиксация изменений и отправка их в локальный репозиторий с помощью коммита.
4. Слияние локального репозитория с оригиналом на сервере.

5. Опишите порядок работы с общим хранилищем VCS.

В общих чертах это: 1. Клонирование исходного репозитория. 2. Внесение изменений в собственную копию. 3. Отправка изменений в основной репозиторий.

6. Каковы основные задачи, решаемые инструментальным средством git?

1. Обеспечение удобной командной работы над общим кодом.
2. Хранение информации обо всех изменениях в коде.

7. Назовите и дайте краткую характеристику командам git.

- **git init**: создание основного дерева репозитория

- **git pull:** получение обновлений (изменений) текущего дерева из центрального репозитория
- **git push:** отправка всех произведённых изменений локального дерева в центральный репозиторий
- **git status:** просмотр списка изменённых файлов в текущей директории
- **git diff:** просмотр текущих изменений:
- **git add:** сохранение текущих изменений:
- **git rm:** удалить указанные файлы из индекса репозитория
- **git commit:** сохранение текущих изменений
- **git checkout:** создание и/или переключение на новую ветку
- **git push:** отправка изменений конкретной ветки в центральный репозиторий
- **git merge:** слияние ветки с текущим деревом
- **git branch d/D/-d:** удаление ветки

8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

При работе с локальным репозиторием мы изменяем код и фиксируем изменения с помощью команд `add` и `commit`. При работе с удалённым репозиторием мы сначала должны клонировать себе копию интересующей нас ветки, затем также использовать `add` и `commit` для фиксации изменений, после чего отправить их на сервер командой `push`.

9. Что такое и зачем могут быть нужны ветви (branches)?

Ветви нужны для разделение работы на несколько частей между несколькими людьми. Например для каждой новой функции можно создать свою ветку, после чего все такие ветки сливаются с основной веткой разработки, которая в свою очередь сливается в ветку подготовки релиза, где происходят только изменения документации и комментариев. Параллельно со всем этим может существовать ветка исправления ошибок, позволяющая делать багфикс не отвлекаясь на код разрабатываемых функций.

10. Как и зачем можно игнорировать некоторые файлы при `commit`?

Игнорировать некоторые файлы можно с помощью команды **git rm** или файла **.gitignore**. Файлы, не относящиеся к коду программ нет смысла сохранять в системе контроля версий. К таким файлам могут относиться исполняемые файлы программ или файлы с тестовыми данными.

Вывод

Я систематизировал свои знания по написанию документов (в частности отчетов по лабораторным работам) в формате Markdown. Все поставленные цели были достигнуты.