Лабораторная работа №2

Научное программирование

Леонтьева Ксения Андреевна | НПМмд-02-23

Содержание

# 1 Цель работы

Повторить процесс оформления отчётов с помощью легковесного языка разметки Markdown.

# 2 Теоретическое введение

**Markdown** - это язык разметки, используемый для форматирования документов всех типов, созданный Джоном Грубером и Аароном Шварцем в 2004 году, сегодня это один из самых популярных языков среди программистов. Многие идеи языка были позаимствованы из существующих соглашений по разметке текста в электронных письмах. Реализации языка Markdown преобразуют текст в формате Markdown в валидный, правильно построенный XHTML и заменяют левые угловые скобки («<») и амперсанды («&») на соответствующие коды сущностей. Первой реализацией Markdown стала написанная Грубером реализация на Perl, однако спустя некоторое время появилось множество реализаций от сторонних разработчиков. Реализация на Perl распространяется по лицензии типа BSD. Реализации Markdown на различных языках программирования включены (или доступны в качестве плагина) во многие системы управления содержимым.

Более подробно см. в [1]

# 3 Выполнение лабораторной работы

В данном разделе представлен отчет по предыдущей лабораторной работе.

Для начала была создана учетная запись и заполнены основные данные на сайте https://github.com (рис. fig. 1).

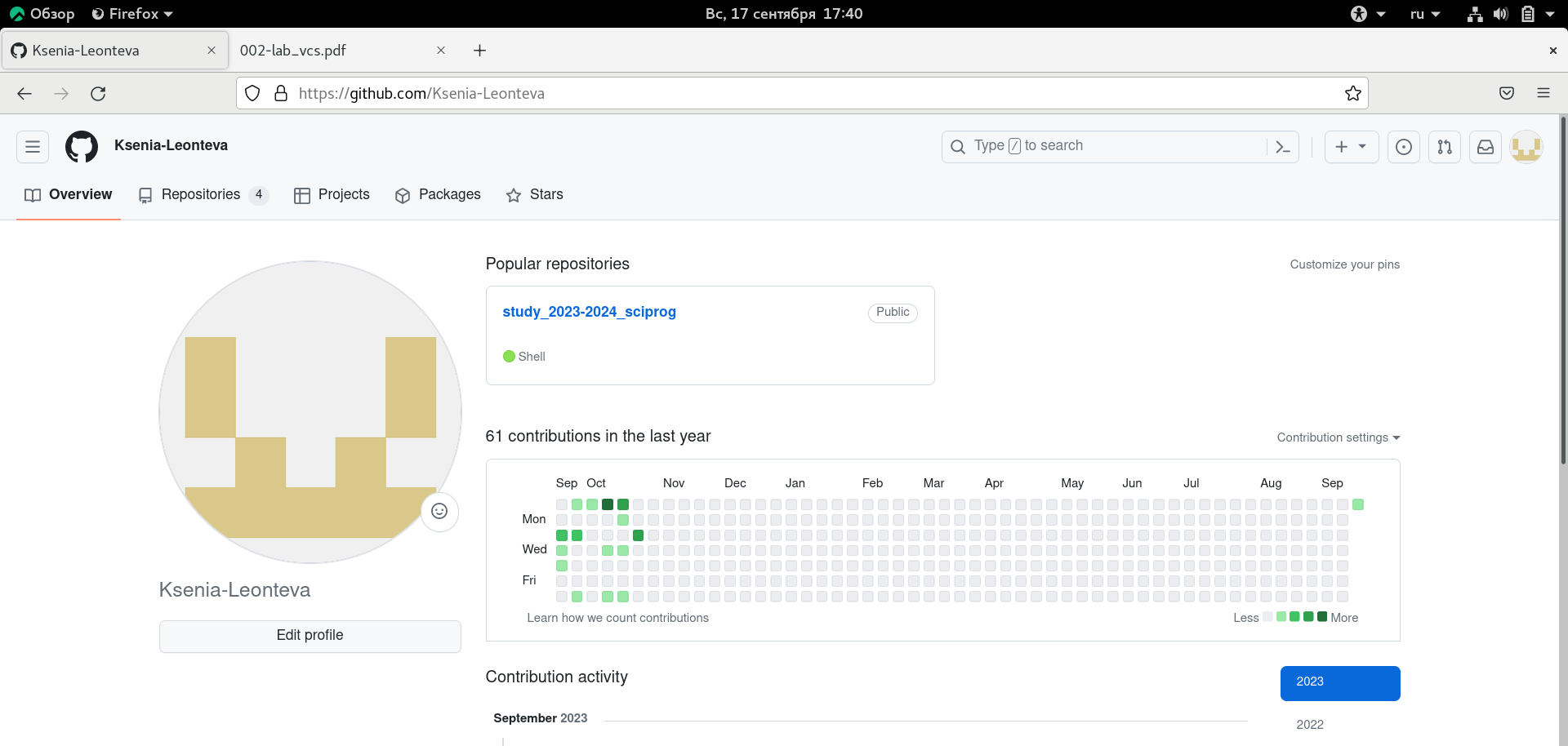


Рис. 1: Учетная запись на GitHub

Вручную установили программное обеспечение git-flow (рис. fig. 2).

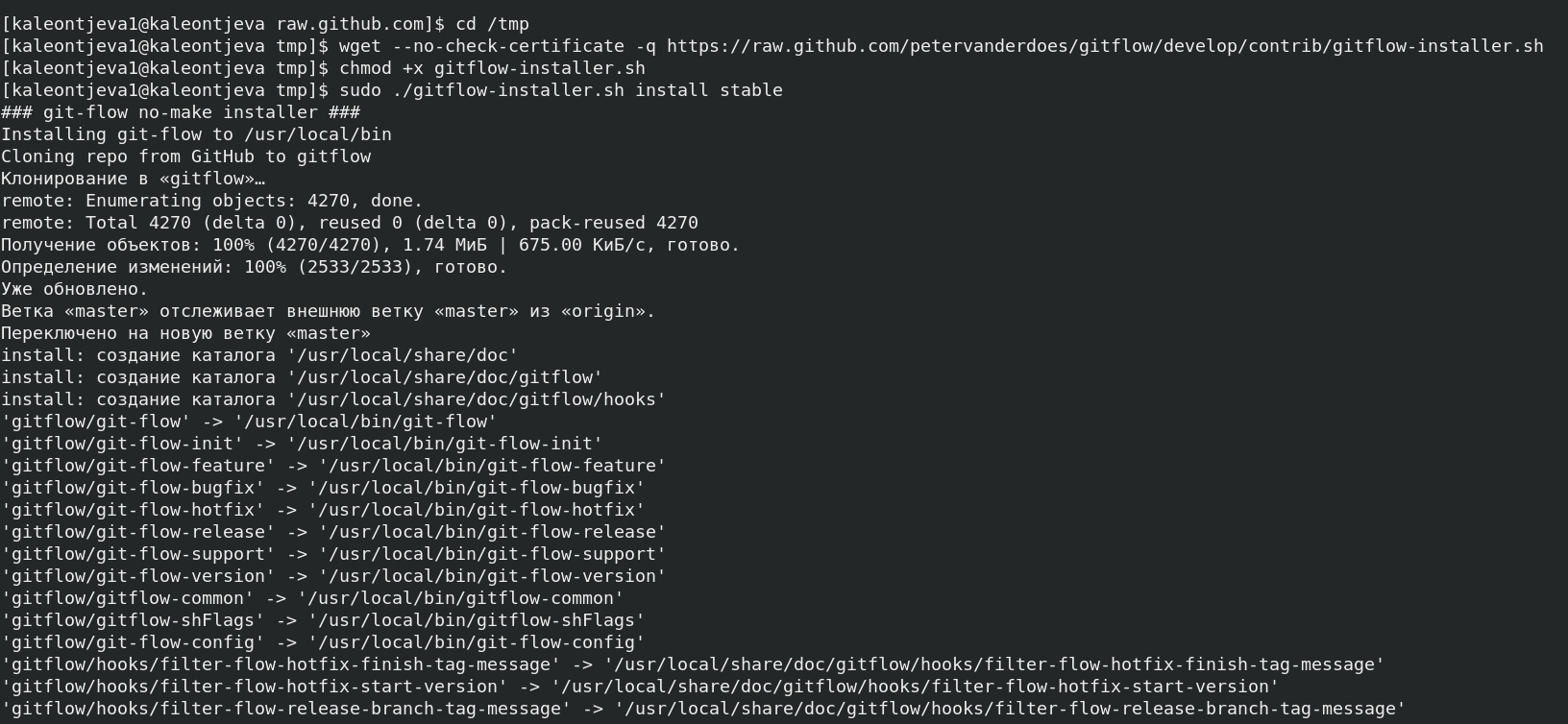


Рис. 2: Установка git-flow

Осуществили базовую настройку git (рис. fig. 3), а именно:

* задали имя и email владельца репозитория,
* настроили utf-8 в выводе сообщений git,
* настроили верификацию и подписание коммитов git,
* задали имя начальной ветки (master),
* параметр autocrlf,
* параметр safecrlf.

Также создали ключи ssh (рис. fig. 3 и рис. fig. 4):

* по алгоритму rsa с ключом размером 4096 бит,
* по алгоритму ed25519.

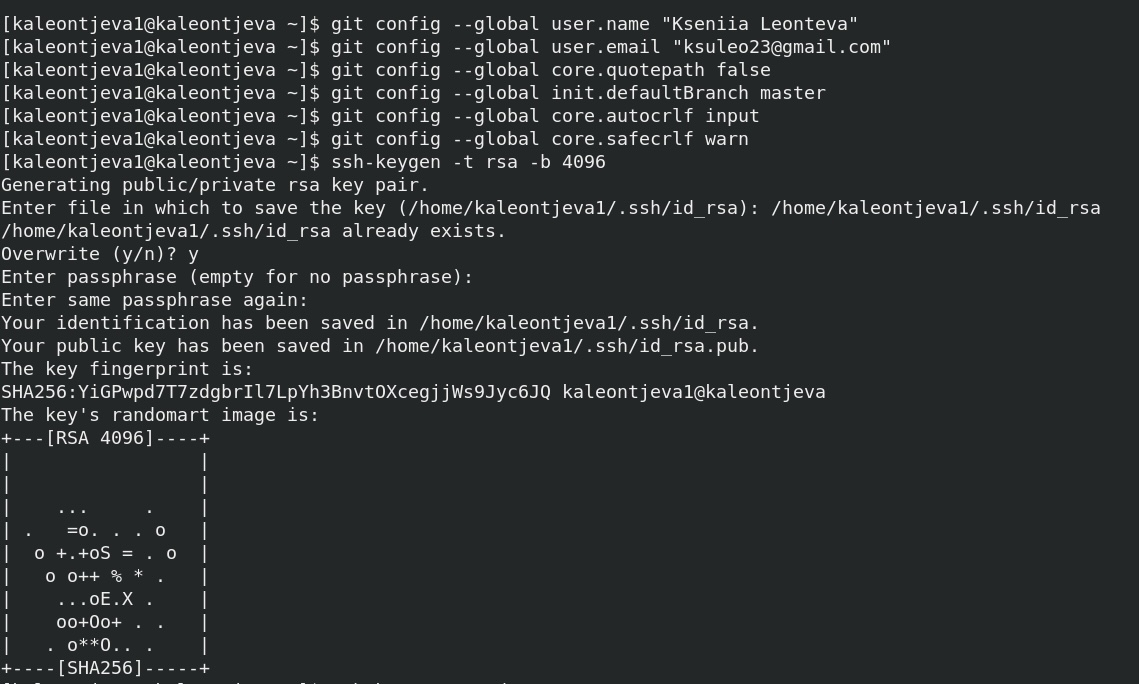


Рис. 3: Базовая настройка git и создание ключа ssh по алгоритму rsa

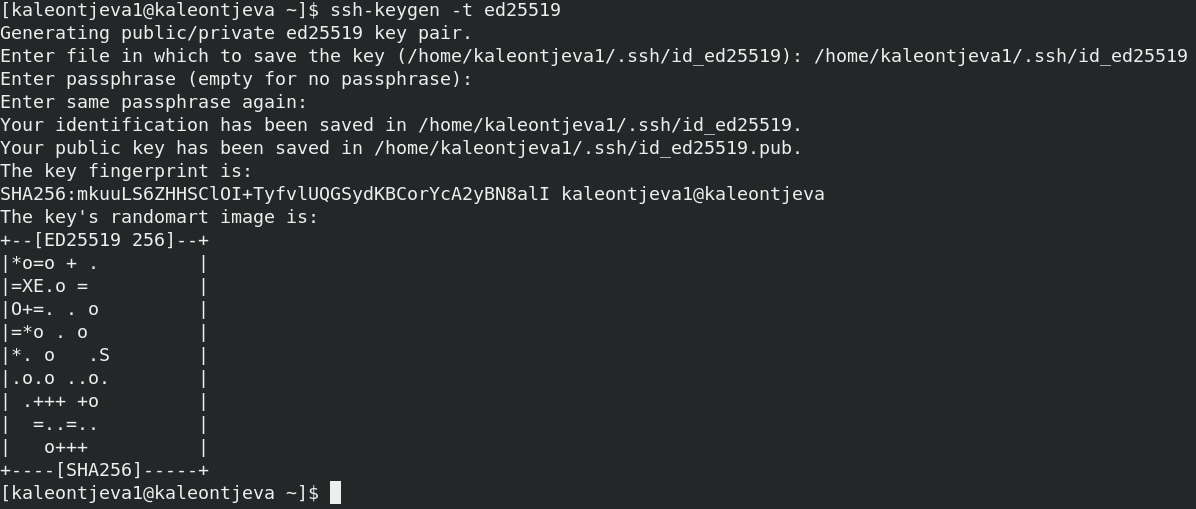


Рис. 4: Создание ключа ssh по алгоритму ed25519

Сгенерировали ключ pgp и выбрали из предложенных опций необходимые (рис. fig. 5 и рис. fig. 6).

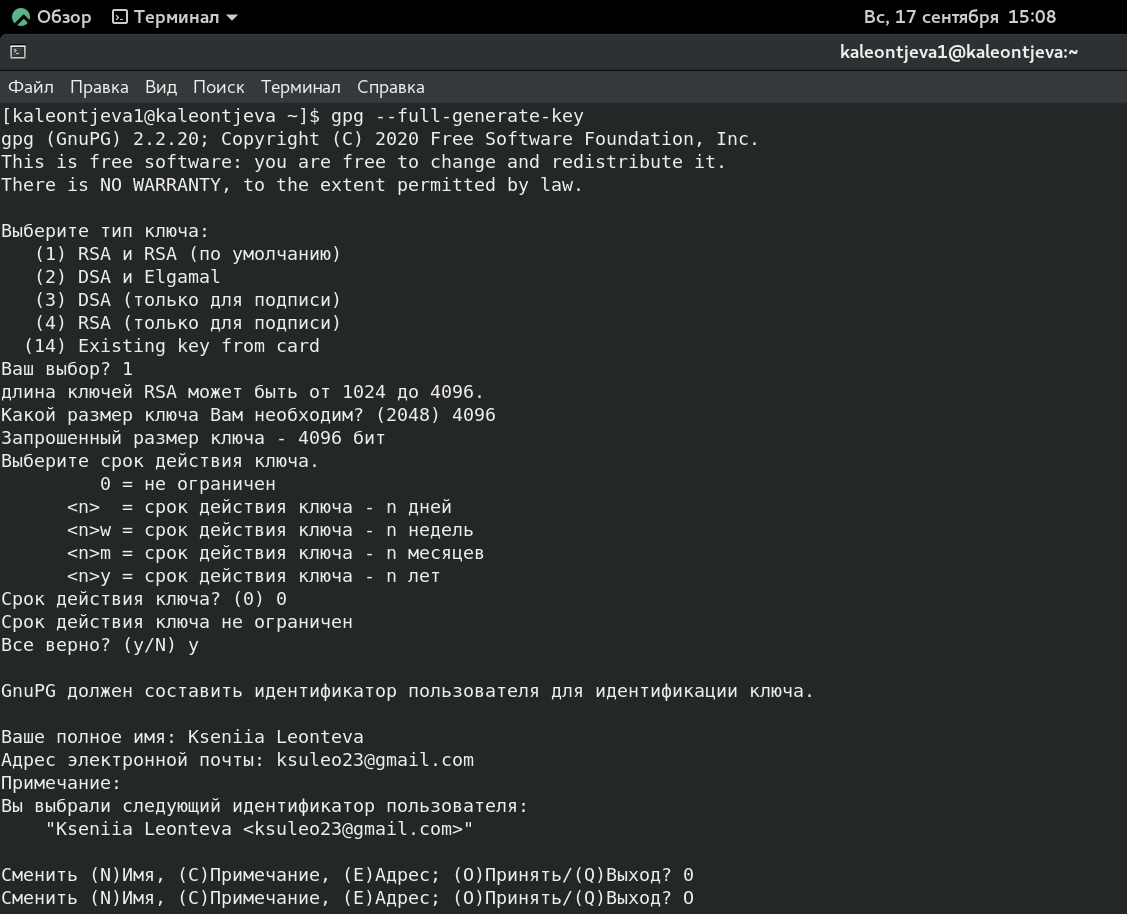


Рис. 5: Генерация ключа pgp

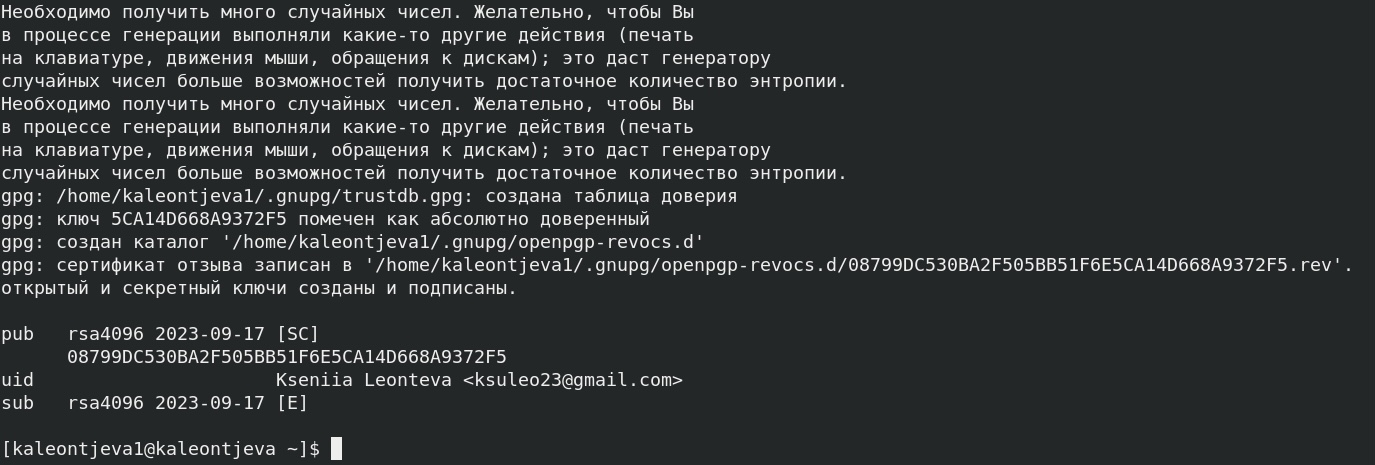


Рис. 6: Генерация ключа pgp

Выводим список ключей и копируем отпечаток приватного ключа (рис. fig. 7).

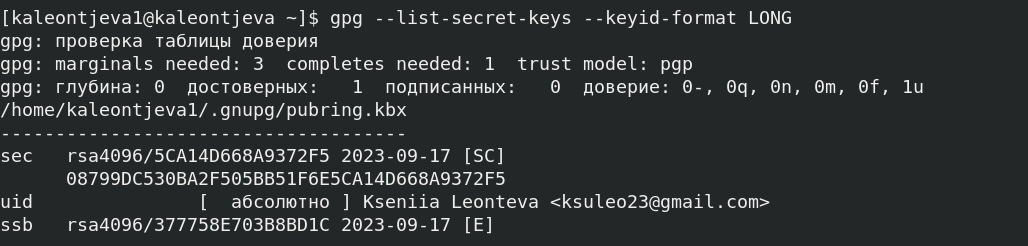


Рис. 7: Вывод списка ключей

Копируем сгенерированный PGP ключ в буфер обмена (рис. fig. 8).

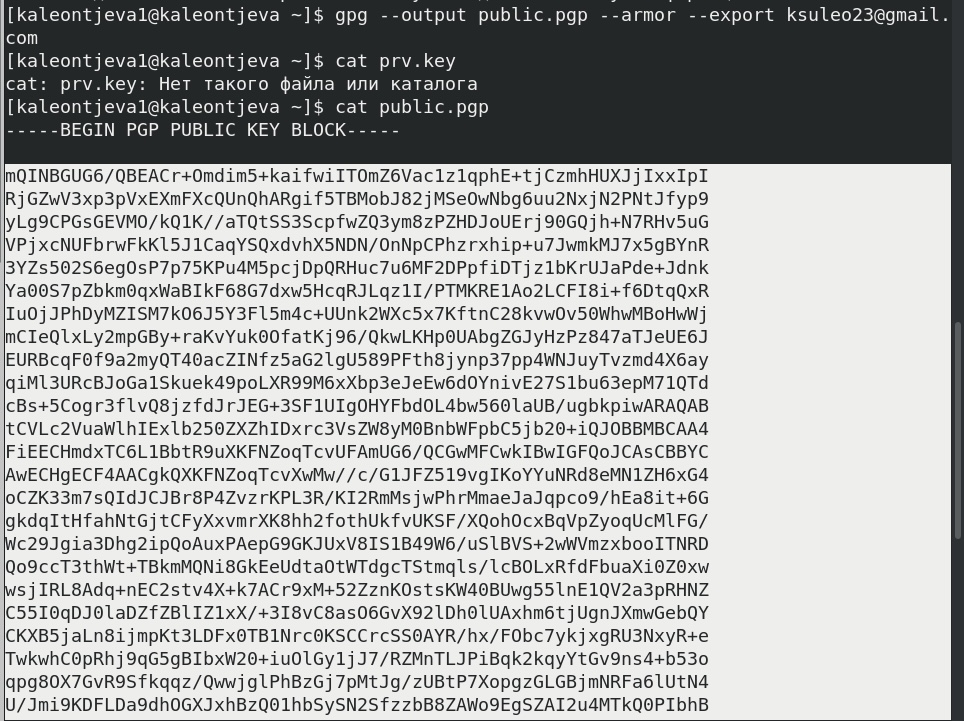


Рис. 8: Копирование сгенерированного ключа PGP

Затем переходим в настройки GitHub, в раздел keys и, нажав на кнопку New GPG key, вставляем полученный ключ в поле ввода (рис. fig. 9).

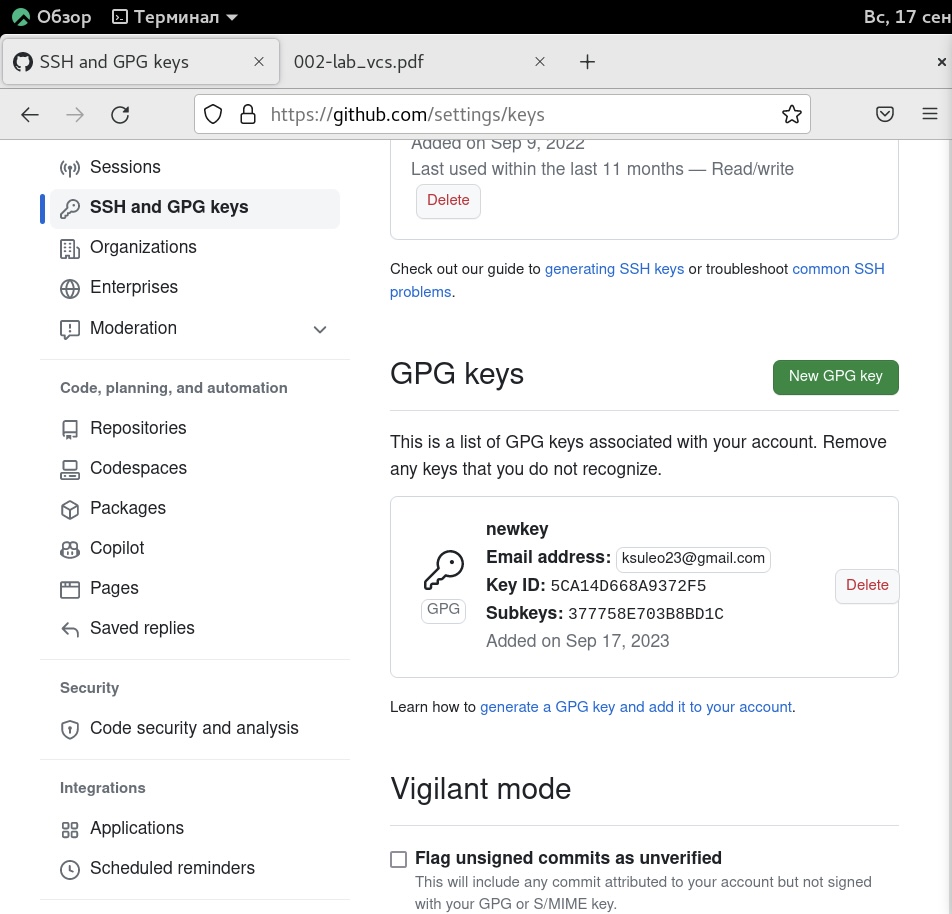


Рис. 9: Добавление gpg-ключа на GitHub

Настраиваем автоматические подписи коммитов git. Используя введенный email, указываем Git поменять его при подписи коммитов (рис. fig. 10).

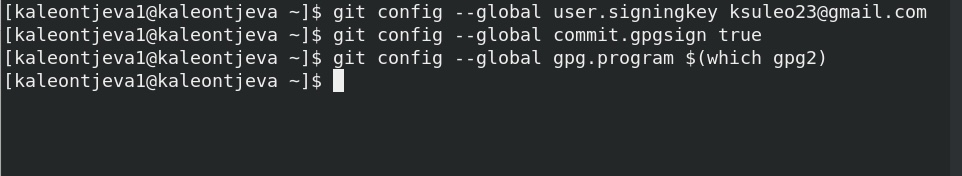


Рис. 10: Настройка автоматических подписей коммитов git

Устанавливаем и настраиваем gh. Отвечаем на наводящие вопросы утилиты и авторизуемся через браузер (рис. fig. 11 - рис. fig. 13).

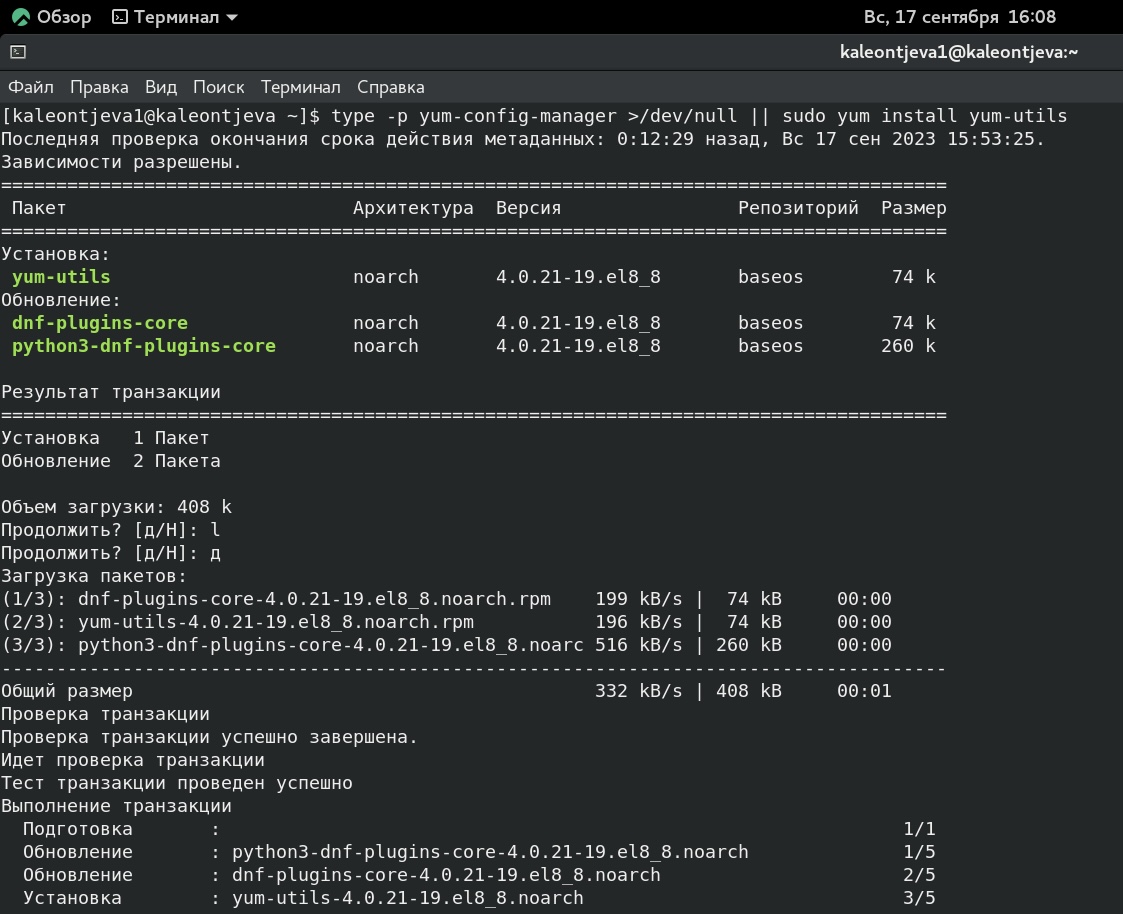


Рис. 11: Установка gh

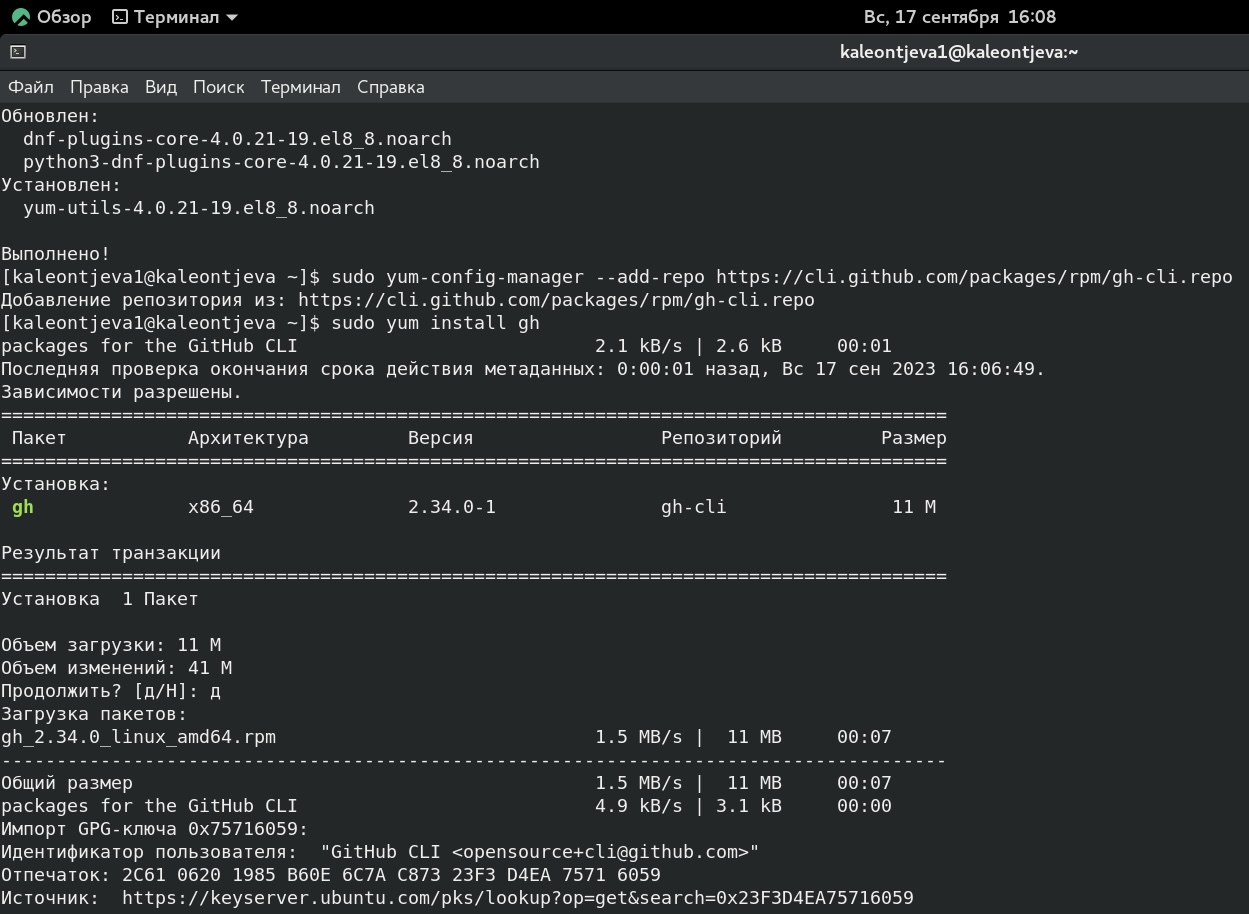


Рис. 12: Установка gh

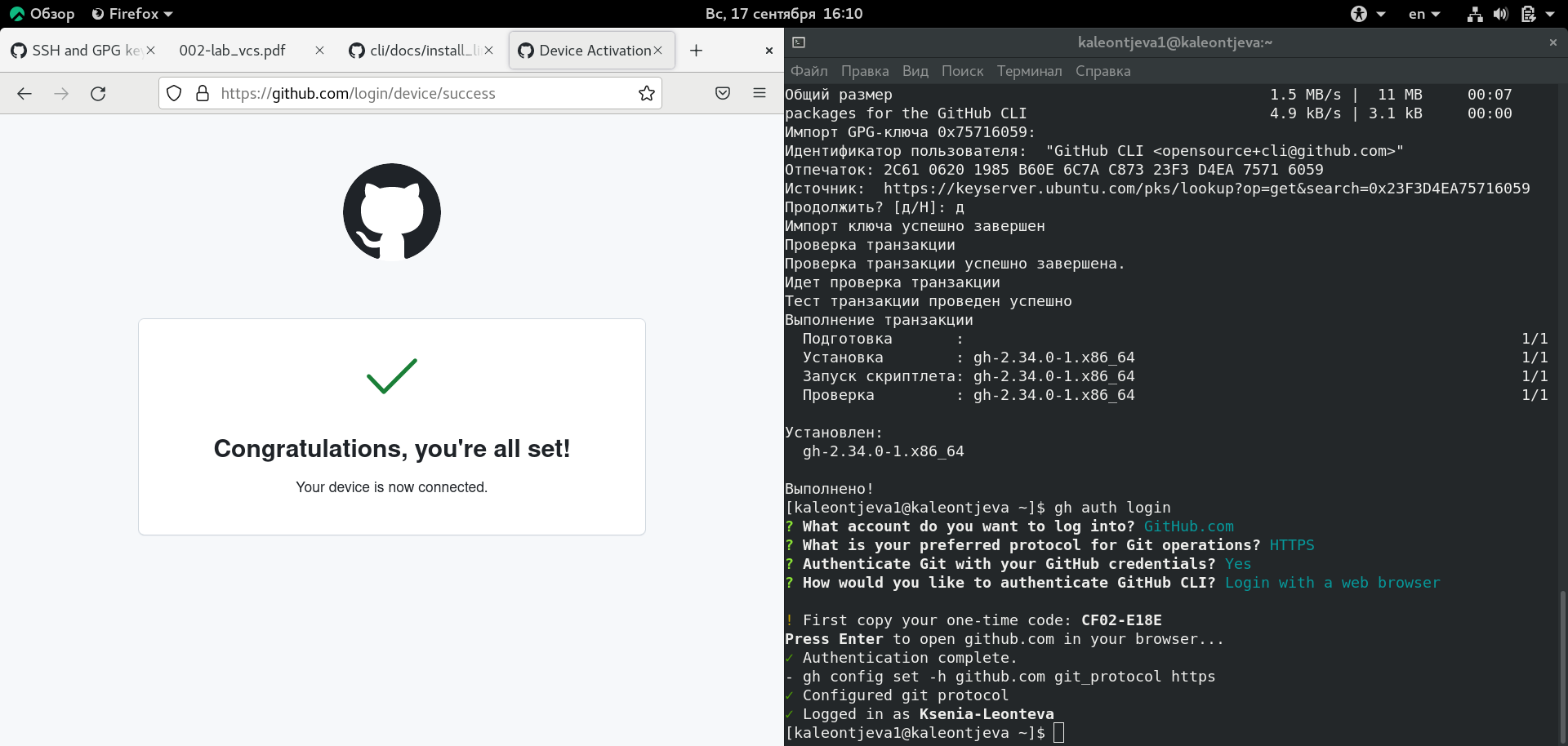


Рис. 13: Настройка gh и авторизация

Создаем репозиторий курса на основе шаблона (рис. fig. 14), параллельно сгенерировав и добавивив на GitHub еще один ssh-ключ (рис. fig. 15 и рис. fig. 16).

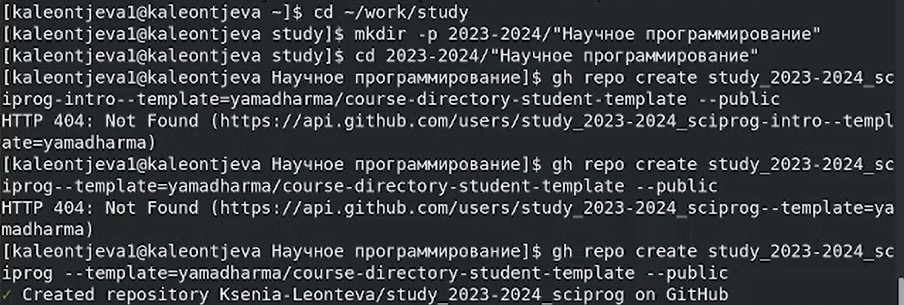


Рис. 14: Создание репозитория курса

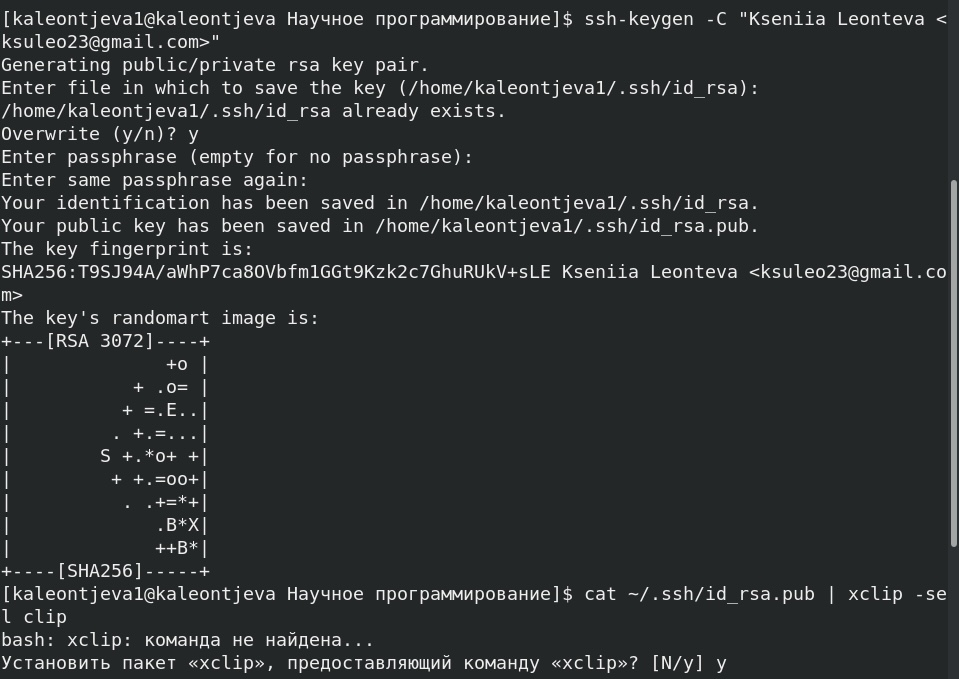


Рис. 15: Генерация ssh-ключа

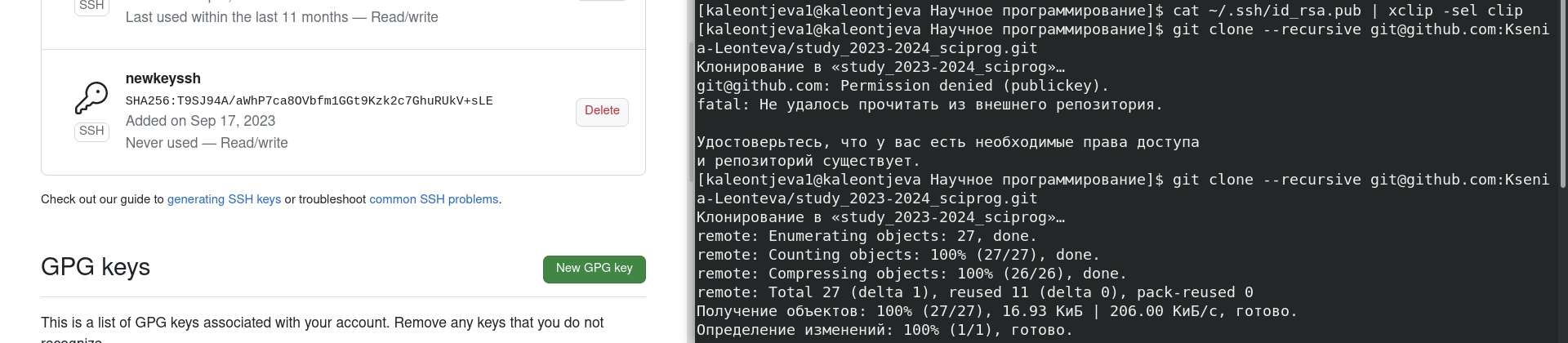


Рис. 16: Добавление ssh-ключа на GitHub и завершение создания репозитория курса

Настраиваем каталог курса (рис. fig. 17 и рис. fig. 18):

* переходим в каталог курса,
* удаляем лишние файлы,
* создаем необходимые каталоги,
* отправляем файлы на сервер.

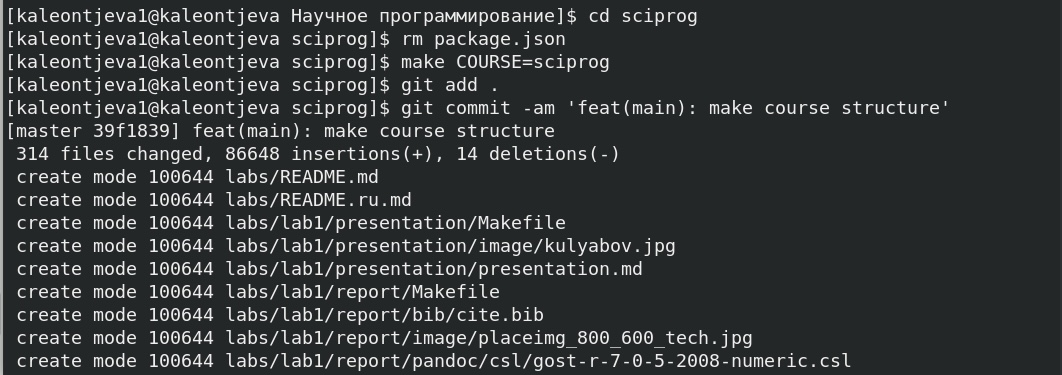


Рис. 17: Настройка каталога курса

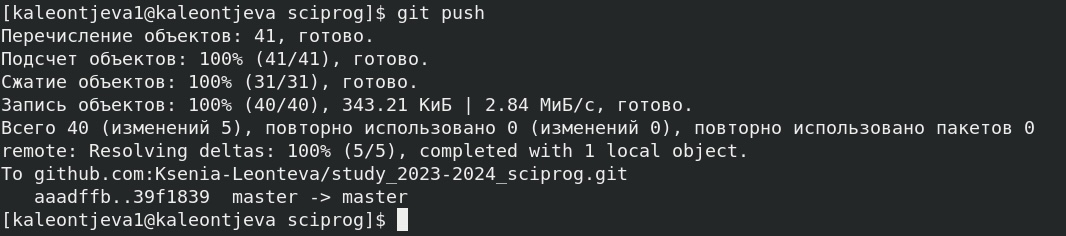


Рис. 18: Настройка каталога курса

# 4 Контрольные вопросы

1. Система контроля версий (Version Control System, VCS) — это программное обеспечение, которое помогает отслеживать изменения в файловой системе и эффективно управлять версиями файлов и кода в проекте. Она позволяет разработчикам работать над проектами совместно, отслеживать, комментировать и объединять свои изменения.
2. Хранилище — место, куда помещается документ после внесения в него нужных правок. Оно является местом хранения служебной информации и всех версий документов. Commit — это пакет изменений, хранящий информацию с добавленными, отредактированными или удаленными файлами (в Git — это команда для записи индексированных изменений в репозиторий Git). История (в Git) — точный реестр всех коммитов, содержащих произведенные с файлами изменения. Рабочая копия — текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.
3. 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.

Децентрализованные (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”. Две наиболее известные DVCS – это Git, Bazaar, Mercurial.

В отличие от централизованных, в распределенных системах контроля версий центральный репозиторий не является обязательным.

1. Создание локального репозитория. Делается предварительная конфигурация:

git config –global user.name “Имя Фамилия”

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

Настраивается 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

1. 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.

1. Основные задачи, решаемые инструментальным средством git: обеспечение удобной командной работы над проектом и хранение информации о всех изменениях в проекте.
2. Наиболее часто используемые команды 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 изменения,
* git add . – добавить все изменённые и/или созданные файлы и/или каталоги,
* git add имена\_файлов – добавить конкретные изменённые и/или созданные файлы и/или каталоги,
* git rm имена\_файлов – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории),
* git commit -am ‘Описание коммита’ – сохранить все добавленные изменения и все изменённые файлы,
* git commit – сохранить добавленные изменения с внесением комментария через встроенный редактор,
* git checkout -b имя\_ветки – создание новой ветки, базирующейся на текущей,
* git checkout имя\_ветки – переключение на некоторую ветку,
* git push origin имя\_ветки – отправка изменений конкретной ветки в центральный репозиторий,
* git merge –no-ff имя\_ветки – слияние ветки с текущим деревом,
* git branch -d имя\_ветки – удаление локальной уже слитой с основным деревом ветки,
* git branch -D имя\_ветки – принудительное удаление локальной ветки,
* git push origin :имя\_ветки – удаление ветки с центрального репозитория.

1. Создание тестового файла hello.txt и добавление его в локальный репозиторий:

echo ‘hello world’ > hello.txt

git add hello.txt

git commit -am ‘Новый файл’

1. Ветки нужны для того, чтобы при работе над проектом программисты работали незваисимо друг от друга. В Git ветки — это элемент повседневного процесса разработки. По сути ветки в Git представляют собой указатель на снимок изменений. Если нужно добавить новую возможность или исправить ошибку (незначительную или серьезную), создается новая ветка, в которой будут размещаться эти изменения.
2. 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. Игнорируемые файлы отслеживаются в специальном файле .gitignore, который регистрируется в корневом каталоге репозитория. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл .gitignore, чтобы указать в нем новые файлы, которые должны быть проигнорированы. Файлы .gitignore содержат шаблоны, которые сопоставляются с именами файлов в репозитории для определения необходимости игнорировать эти файлы.

# 5 Вывод

В ходе выполнения данной лабораторной работы я повторила процесс оформления отчётов с помощью легковесного языка разметки Markdown.

# Список литературы

1. Markdown [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Markdown>.