**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей**

Отчет

По лабораторной работе №2

*Дисциплина: операционные системы*

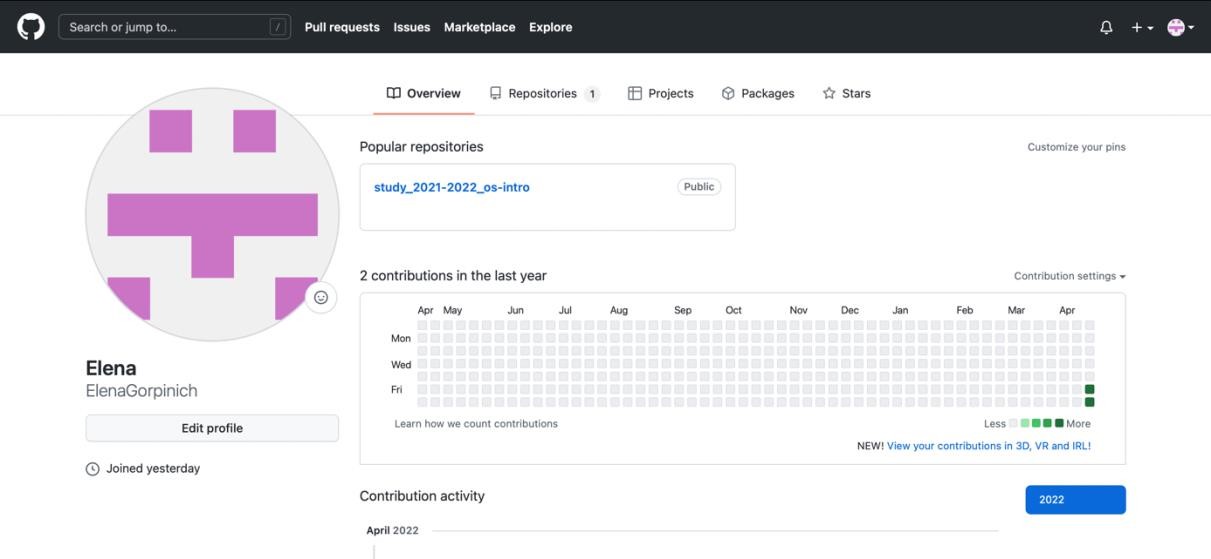
Студентка: Горпинич Елена Михайловна

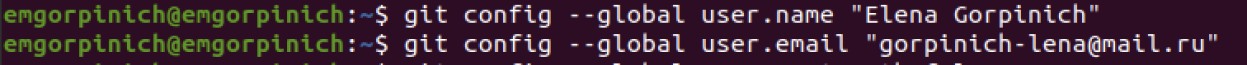
Группа: НПМбд-01–21

**Москва 2022**

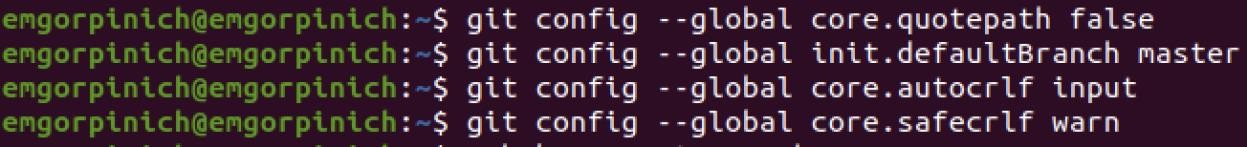
Цель работы: изучить идеологию и применение средств контроля версий и освоить умения по работе с git.

Ход работы:

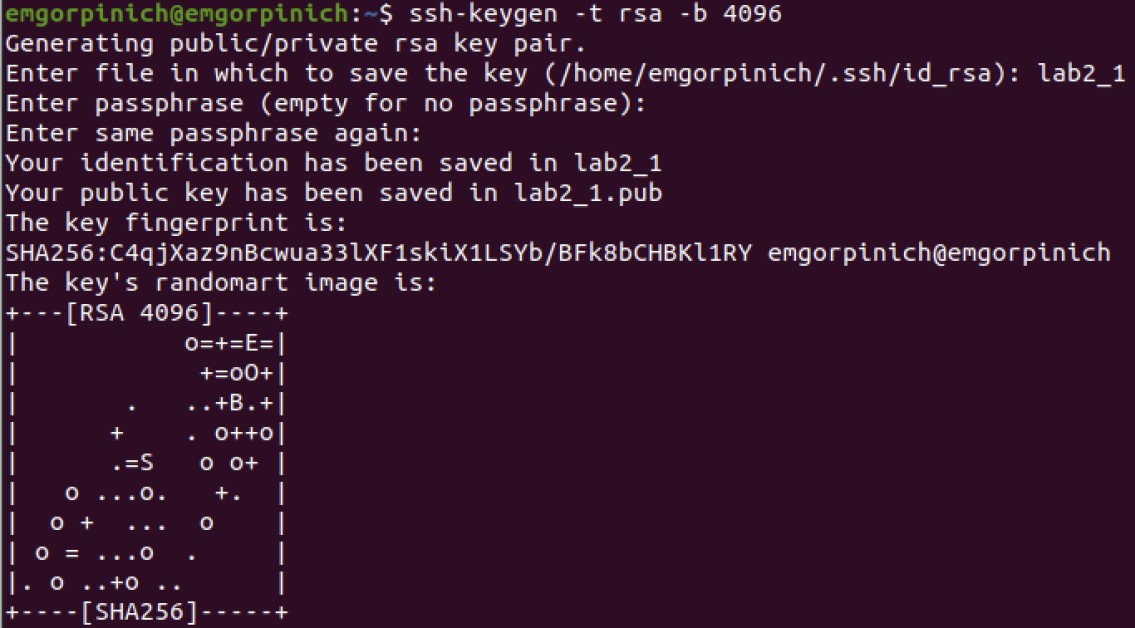
1. Настройка GitHub. Изначально создаём учетную запись и заполняем основные данные на официальном сайте. [https://github.com](https://github.com/).
2. Затем синхронизируем учетную запись с компьютером с помощью терминала.



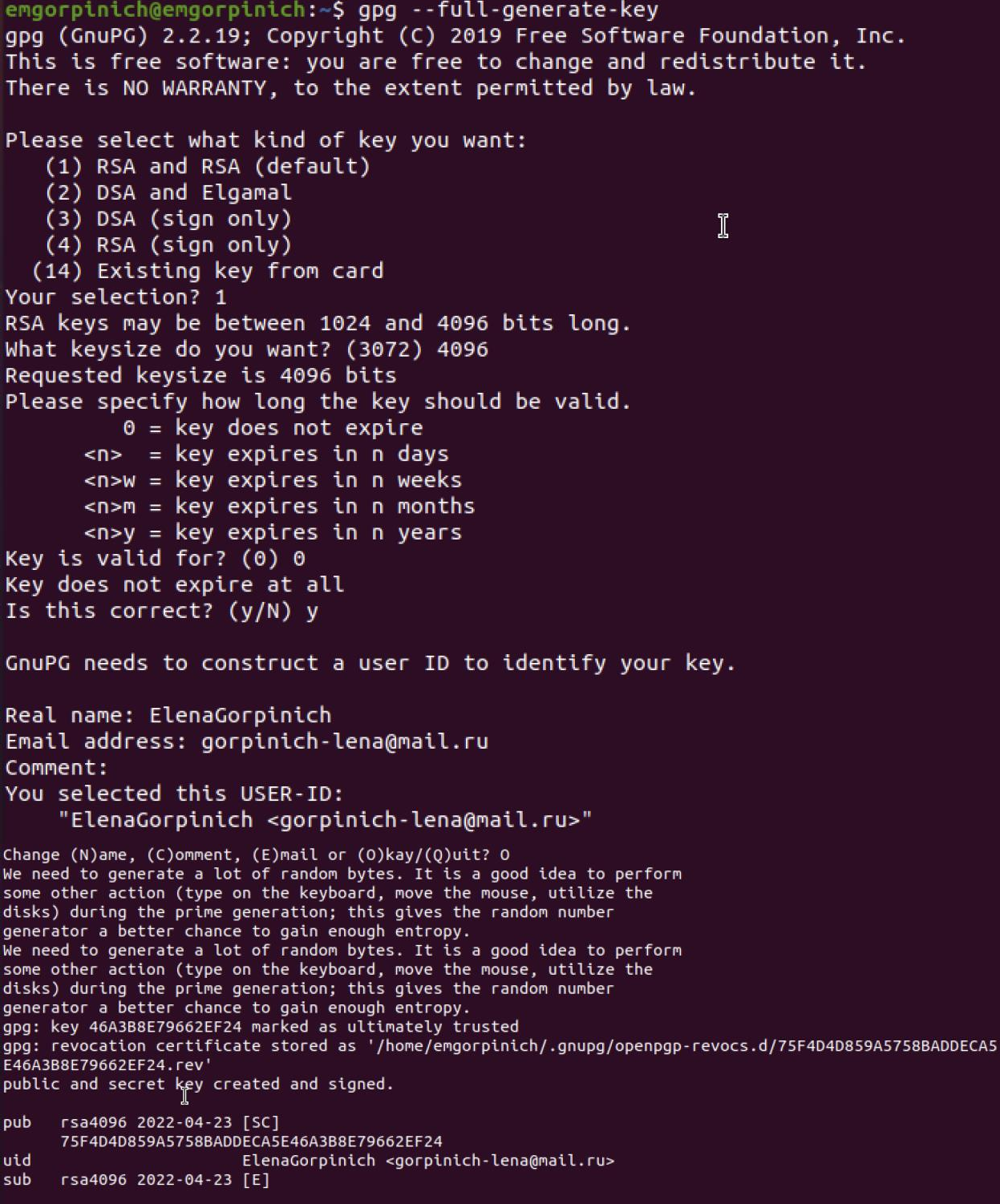
1. Начинаем базовую настройку git: настроим utf-8 в выводе сообщений git, верификацию и подписание коммитов git, зададим имя начальной̆ ветки (будем называть её master), параметр autocrlf, параметр safecrlf



1. Создадим ключ ssh по алгоритму rsa с ключевым размером 4096 бит

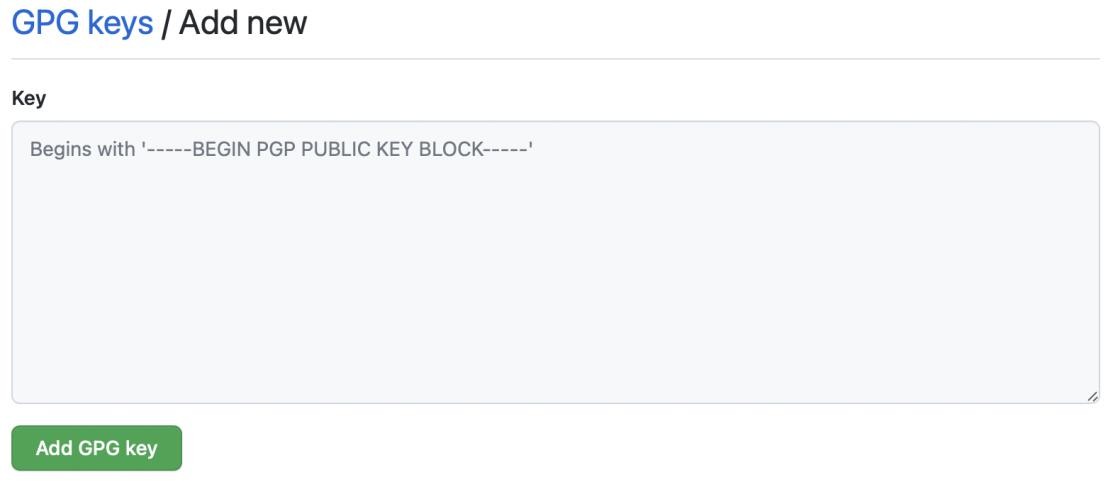
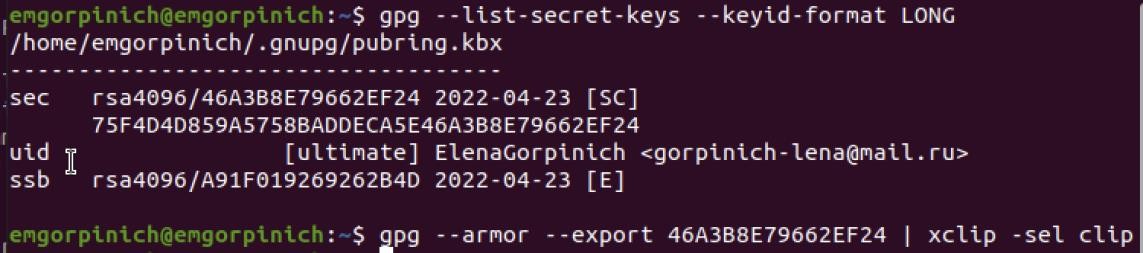


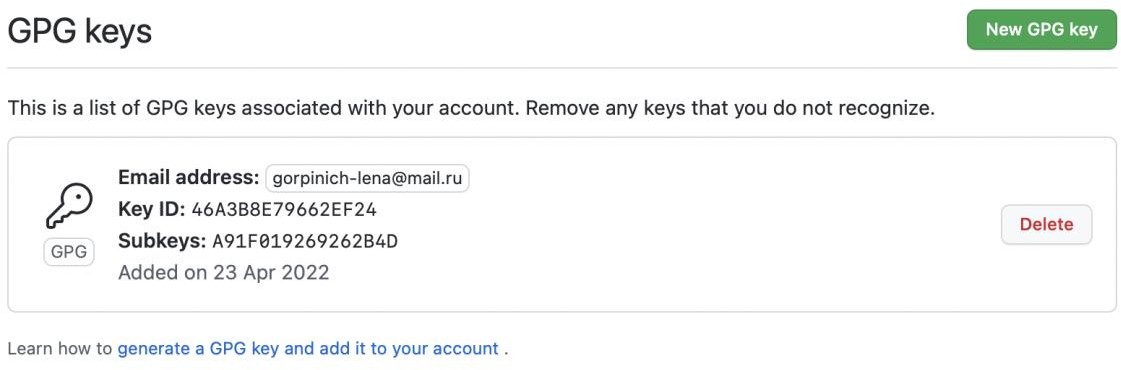
1. Создадим ключ pgp. Генерируем ключ и из предложенных опций выбираем опции описанные в условии лабораторной работы.



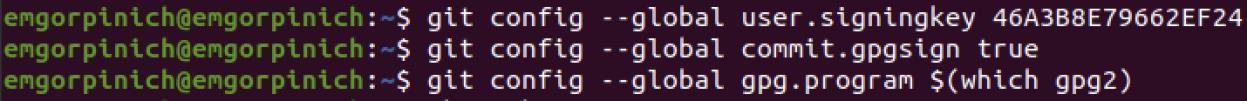
1. Добавим PGP ключ в GitHub. Для этого выводим список ключей и скопируем отпечаток приватного ключа. Скопируем наш

сгенерированный PGP ключ в буфер обмена. С помощью настроек GitHub (https://github.com/settings/keys), добавим наш скопированный ключ, вставив его в поле ввода, которое появилось после нажатия кнопки New GPG key



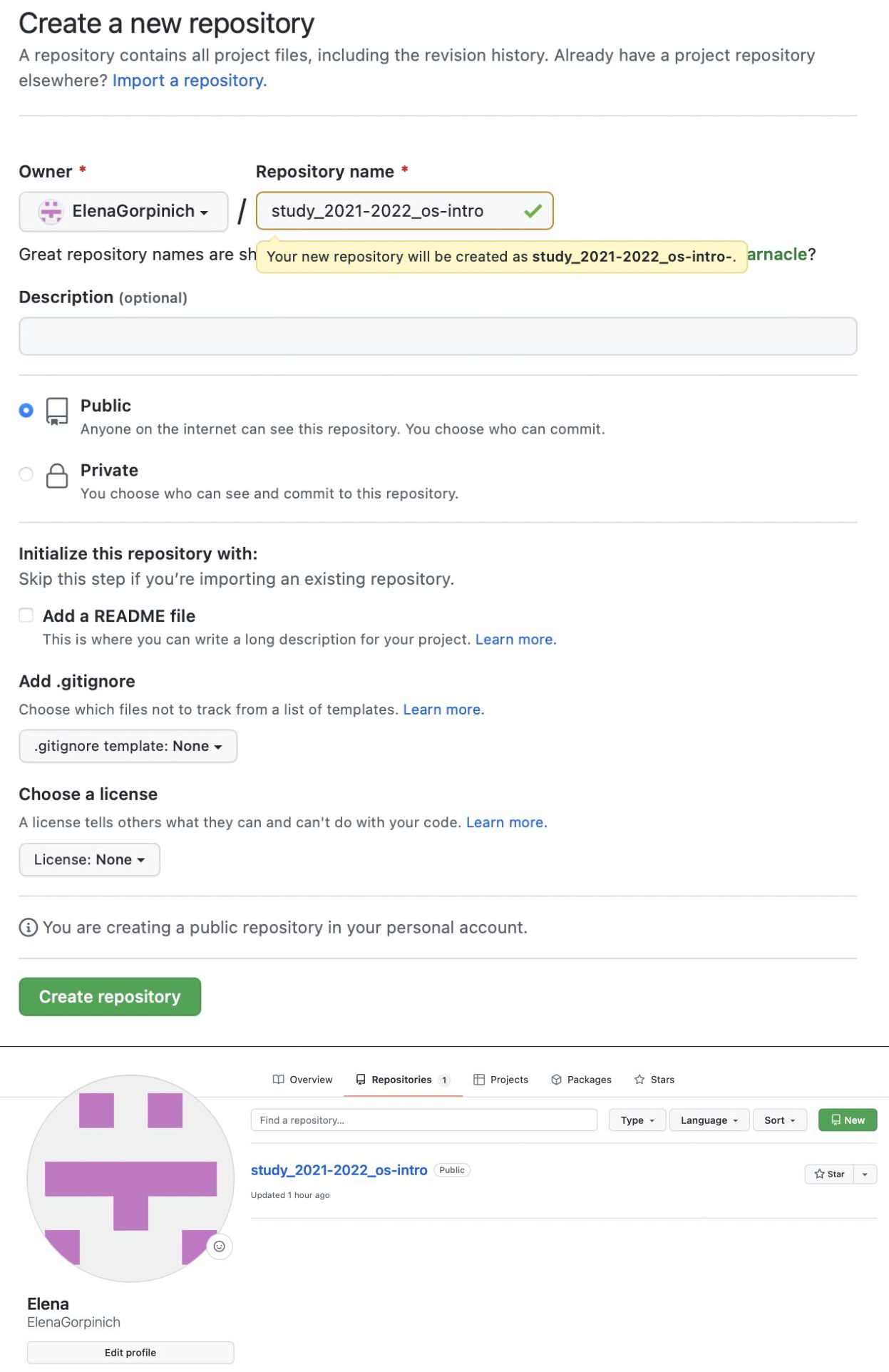


1. Настроим автоматических подписей коммитов git



1. Создадим новый репозиторий курса на основе шаблона и далее

настроим каталог курса, удалив ненужные файлы, создав необходимые каталоги и отправив файлы на сервер



Вывод:

В данной лабораторной работе я научилась работать с Github (создавать и привязывать учетную запись к компьютеру). Разобрала основные команды git и рассмотрела, как их применять их при работе с Github. Изучила идеологию и научилась применять средства контроля версий

Контрольные вопросы:

* VCS (Система контроля версий) — 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.

* Терминология VCS:
  + Репозиторий - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией.
  + Commit сохранение изменений в репозитории
  + Рабочая копия - копия проекта, связанная с репозиторием
* *Централизованные VCS*

Одно основное хранилище всего проекта

Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно (Subversion, CVS, TFS, VAULT, AccuRev)

*Децентрализованные VCS*

У каждого пользователя свой вариант (возможно, не один) репозитория Присутствует возможность добавлять и забирать изменения из любого репозитория. (Git, Mercurial, Bazaar)

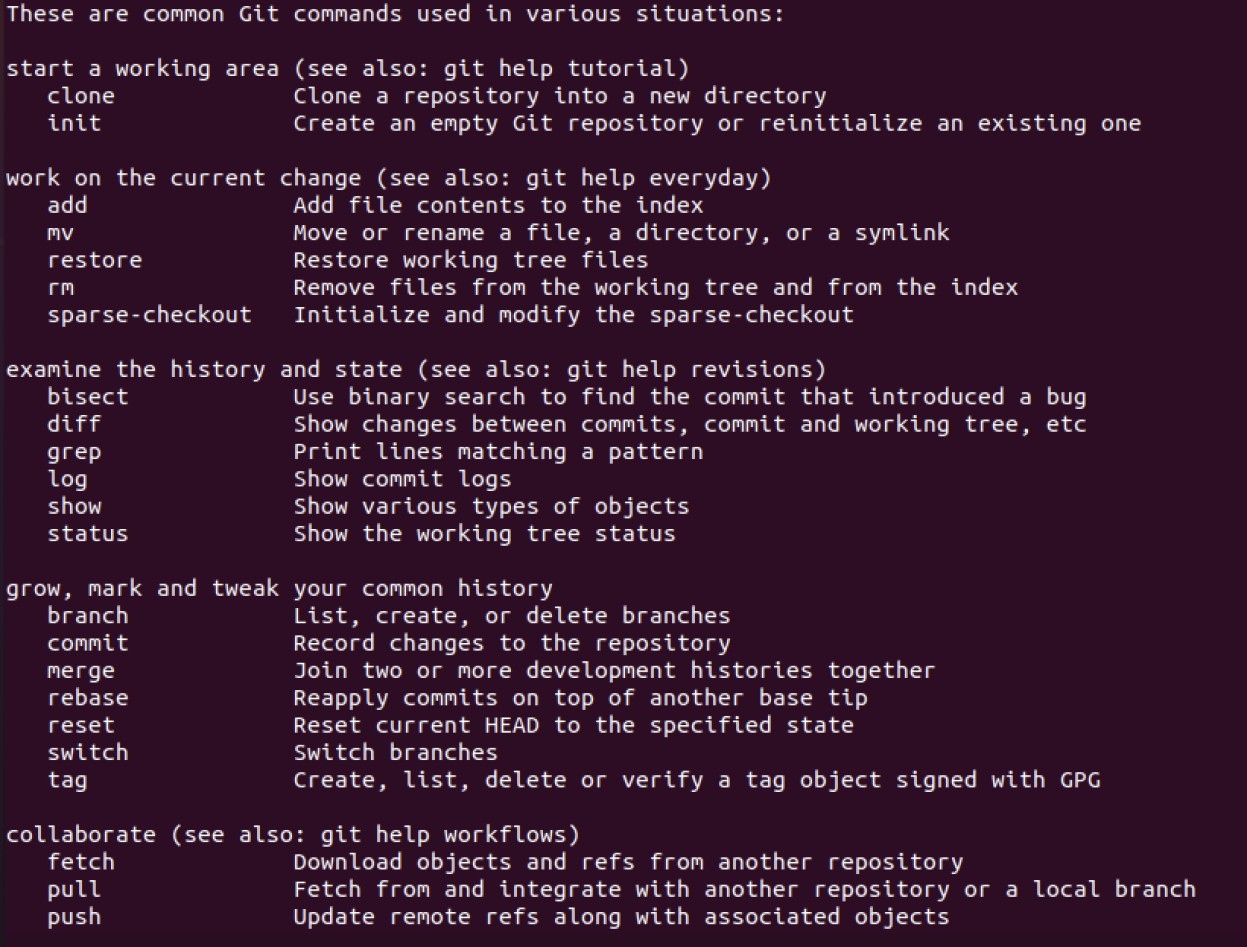
* Поддержка автономной работы; локальные фиксации изменений могут быть отправлены позже

Каждое рабочее дерево в git содержит хранилище с полной историей проекта

Ни одно хранилище не является по своей природе более важным, чем любое другое

Быстрое и легкое ветвление

* При выполнении команды git –help можно увидеть список команд git, используемых в различных ситуациях



* git fetch

Команда git fetch связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально.

Мы познакомились с ней в разделе [Получение изменений из удалённого](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) [репозитория — Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) главы 2 и продолжили знакомство в

разделе [Удалённые ветки](https://git-scm.com/book/ru/v2/ch00/r_remote_branches) главы 3.

Мы использовали эту команду в нескольких примерах из раздела [Участие в проекте](https://git-scm.com/book/ru/v2/ch00/r_contributing_project).

Мы использовали её для скачивания запросов на слияние (pull request) из других репозиториев в разделе [Ссылки на запрос слияния](https://git-scm.com/book/ru/v2/ch00/r_pr_refs) главы 6, также мы рассмотрели использование git fetchдля работы с упакованными репозиториями в разделе [Создание пакетов](https://git-scm.com/book/ru/v2/ch00/r_bundling) главы 7.

Мы рассмотрели тонкую настройку git fetch в главе и [Спецификации](https://git-scm.com/book/ru/v2/ch00/r_refspec) [ссылок](https://git-scm.com/book/ru/v2/ch00/r_refspec).

git pull

Команда git pull работает как комбинация команд git fetch и git merge, т. е. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.

Мы познакомились с ней в разделе [Получение изменений из удалённого](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) [репозитория — Fetch и Pull](https://git-scm.com/book/ru/v2/ch00/r_fetching_and_pulling) главы 2 и показали как узнать, какие

изменения будут приняты в случае применения в разделе [Просмотр](https://git-scm.com/book/ru/v2/ch00/r_inspecting_remote) [удаленного репозитория](https://git-scm.com/book/ru/v2/ch00/r_inspecting_remote) главы 2.

Мы также увидели как она может оказаться полезной для разрешения сложностей при перемещении веток в разделе [Меняя базу, меняй](https://git-scm.com/book/ru/v2/ch00/r_rebase_rebase) [основание](https://git-scm.com/book/ru/v2/ch00/r_rebase_rebase) главы 3.

Мы показали как можно использовать только URL удалённого репозитория без сохранения его в списке удалённых репозиториев в разделе [Извлечение удалённых веток](https://git-scm.com/book/ru/v2/ch00/r_checking_out_remotes) главы 5.

И наконец мы показали как проверять криптографические подписи полученных коммитов, используя опцию --verify-signatures в

разделе [Подпись коммитов](https://git-scm.com/book/ru/v2/ch00/r_signing_commits) главы 7. git push

Команда git push используется для установления связи с удалённым репозиторием, вычисления локальных изменений отсутствующих в нём, и собственно их передачи в вышеупомянутый репозиторий. Этой

команде нужно право на запись в репозиторий, поэтому она использует аутентификацию.

Мы познакомились с этой командой в разделе [Отправка изменений в](https://git-scm.com/book/ru/v2/ch00/r_pushing_remotes) [удаленный репозиторий (Push)](https://git-scm.com/book/ru/v2/ch00/r_pushing_remotes) главы 2. Там мы рассмотрели основы обновления веток в удалённом репозитории. В разделе [Отправка](https://git-scm.com/book/ru/v2/ch00/r_pushing_branches)

[изменений](https://git-scm.com/book/ru/v2/ch00/r_pushing_branches) главы 3 мы подробнее познакомились с этой командой, а в разделе [Отслеживание веток](https://git-scm.com/book/ru/v2/ch00/r_tracking_branches) главы 3 мы узнали как настроить

отслеживание веток для автоматической передачи на удалённый репозиторий. В разделе [Удаление веток на удалённом сервере](https://git-scm.com/book/ru/v2/ch00/r_delete_branches) главы 3 мы использовали флаг --delete для удаления веток на сервере, используя git push.

На протяжении раздела [Участие в проекте](https://git-scm.com/book/ru/v2/ch00/r_contributing_project) мы показали несколько примеров использования git push для совместной работы в нескольких удалённых репозиториях одновременно.

В разделе [Публикация изменений подмодуля](https://git-scm.com/book/ru/v2/ch00/r_publishing_submodules) главы 7 мы использовали опцию --recurse-submodules чтобы удостовериться, что все подмодули будут опубликованы перед отправкой на проекта на сервер, что может быть реально полезным при работе с репозиториями, содержащими подмодули.

В разделе [Прочие хуки на стороне клиента](https://git-scm.com/book/ru/v2/ch00/r_other_client_hooks) главы 8 мы поговорили о триггере pre-push, который может быть выполнен перед отправкой данных, чтобы проверить возможность этой отправки.

Наконец, в разделе [Спецификации ссылок для отправки данных на](https://git-scm.com/book/ru/v2/ch00/r_pushing_refspecs)

[сервер](https://git-scm.com/book/ru/v2/ch00/r_pushing_refspecs) главы 10 мы рассмотрели передачу данных с полным указанием передаваемых ссылок, вместо использования распространённых

сокращений. Это может быть полезным если вы хотите очень точно указать, какими изменениями хотите поделиться.

git remote

Команда git remote служит для управления списком удалённых репозиториев. Она позволяет сохранять длинные URL репозиториев в виде понятных коротких строк, например «origin», так что вам не придётся забивать голову всякой ерундой и набирать её каждый раз для

связи с сервером. Вы можете использовать несколько удалённых репозиториев для работы и git remote поможет добавлять, изменять и удалять их.

Эта команда детально рассмотрена в разделе [Работа с удалёнными](https://git-scm.com/book/ru/v2/ch00/r_remote_repos) [репозиториями](https://git-scm.com/book/ru/v2/ch00/r_remote_repos) главы 2, включая вывод списка удалённых репозиториев, добавление новых, удаление или переименование существующих.

Она используется практически в каждой главе, но всегда в одном и том же виде: git remote add <имя> <URL>.

git archive

Команда git archive используется для упаковки в архив указанных коммитов или всего репозитория.

Мы использовали git archive для создания тарбола (tar.gz файла) всего проекта для передачи по сети в разделе [Подготовка релиза](https://git-scm.com/book/ru/v2/ch00/r_preparing_release) главы 5. git submodule

Команда git submodule используется для управления вложенными репозиториями. Например, это могут быть библиотеки или другие, используемые не только в этом проекте ресурсы. У

команды submodule есть несколько подкоманд — add, update, sync и др. — для управления такими репозиториями.

* Ветвь – независимое направление разработки. Изменения, вносимые в одну ветвь, не влияют на остальные ветви. Ветви используются, например, для внесения изменений, которые могут дестабилизировать код или для поддержки релизов. Изменения могут быть перенесены между ветвями посредством слияния.
* Вы можете заставить Git игнорировать определенные файлы и каталоги, то есть исключить их от отслеживания Git - путем создания одного или нескольких файлов [.gitignore](https://git-scm.com/docs/gitignore) в вашем репозитории.

В проектах программного обеспечения .gitignore обычно содержит список файлов и / или каталогов, которые генерируются во время процесса сборки или во время выполнения. Записи в

файле .gitignore могут включать имена или пути, указывающие на: временные ресурсы, например, кэши, файлы журналов,

скомпилированный код и т. д.

файлы локальной конфигурации, которые не должны использоваться совместно с другими разработчиками

файлы, содержащие секретную информацию, такие как пароли входа, ключи и учетные данные

При создании в каталоге верхнего уровня правила будут применяться рекурсивно ко всем файлам и подкаталогам во всем репозитории. При

создании в подкаталоге правила будут применяться к этому конкретному каталогу и его подкаталогам.

Когда файл или каталог игнорируются, это не будет: отслеживается Git

сообщается командами, такими как git status или git diff с такими командами, как git add -A

В необычном случае, когда вам нужно игнорировать отслеживаемые файлы, следует соблюдать особую осторожность