Отчёт по лабораторной работе №2

Первоначальна настройка git

Ганина Таисия Сергеевна

Содержание

# 1 Цель работы

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

# 2 Задание

1. Базовая настройка git.
2. Создание SSH ключа.
3. Создание PGP ключа.
4. Добавление PGP ключа в GitHub.
5. Настройка автоматических подписей коммитов git.
6. Настройка gh.
7. Шаблон для рабочего пространства.
8. Создание локального репозитория курса на основе шаблона.
9. Настройка каталога курса.
10. Контрольные вопросы.

# 3 Теоретическое введение

## 3.1 Системы контроля версий. Общие понятия

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 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 (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.

Примеры использования 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.
* Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

Например, в табл. [1](#tbl:std-dir) приведено описание основных команд системы git.

Table 1: Описание основных команд системы git

| Команда | Действие |
| --- | --- |
| 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 . / git add <имя файла> / git rm <имя файла> | Сохранение текущих изменений |
| git commit / git commit -am "описание коммита" | Сохранение добавленных изменений |
| 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 |

# 4 Выполнение лабораторной работы

1. Базовая настройка git (рис. [1](#fig:001)).

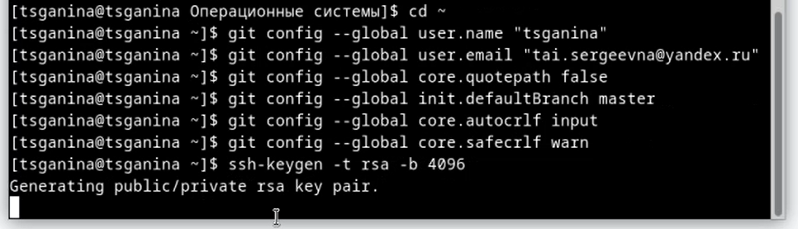


Figure 1: Базовая настройка git

1. Создание SSH ключа (рис. [2](#fig:002), [3](#fig:003)).

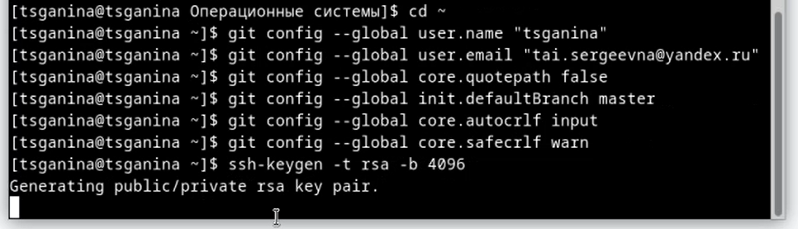


Figure 2: Создание ключа по алгоритму rsa с ключём размером 4096 бит

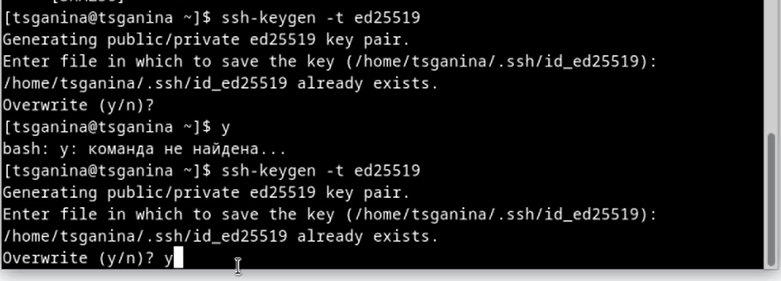


Figure 3: Создание ключа по алгоритму ed25519

1. Создание PGP ключа (рис. [4](#fig:004), [5](#fig:005), [6](#fig:006), [7](#fig:007), [8](#fig:008), [9](#fig:009)).

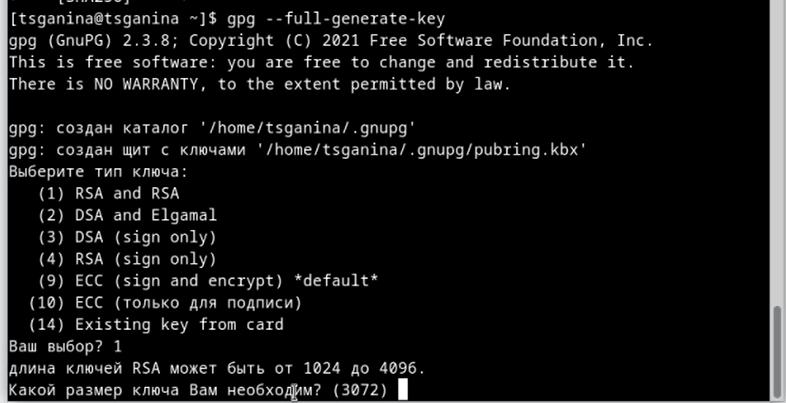


Figure 4: Команда начала генерации ключа

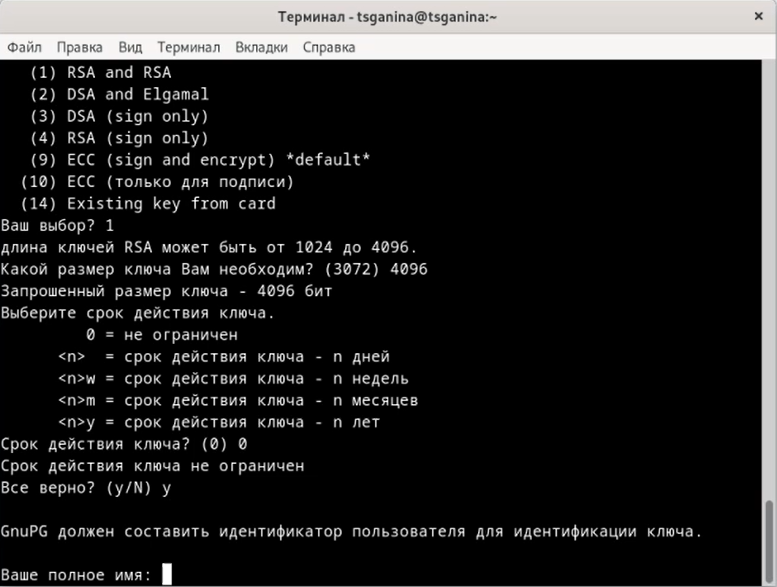


Figure 5: Настройка параметров

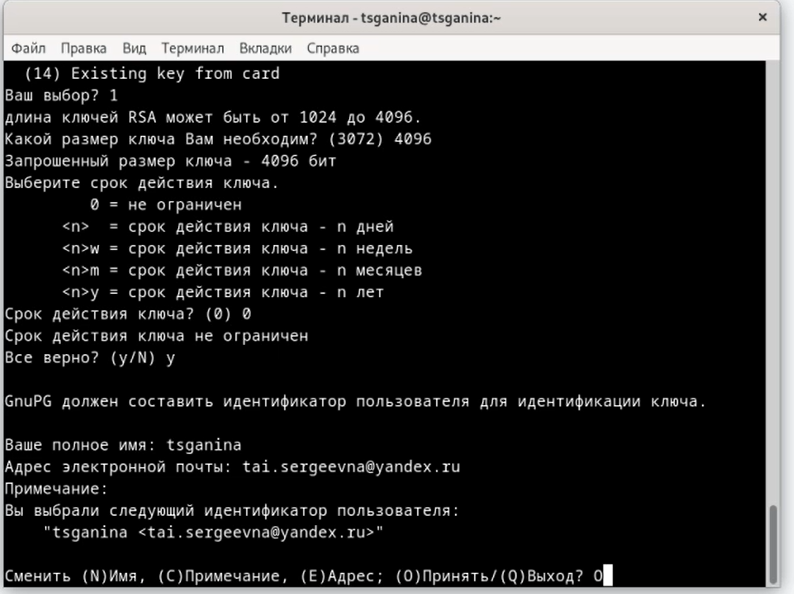


Figure 6: Настройка параметров

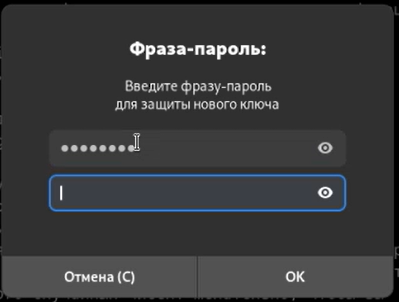


Figure 7: Создание пароля

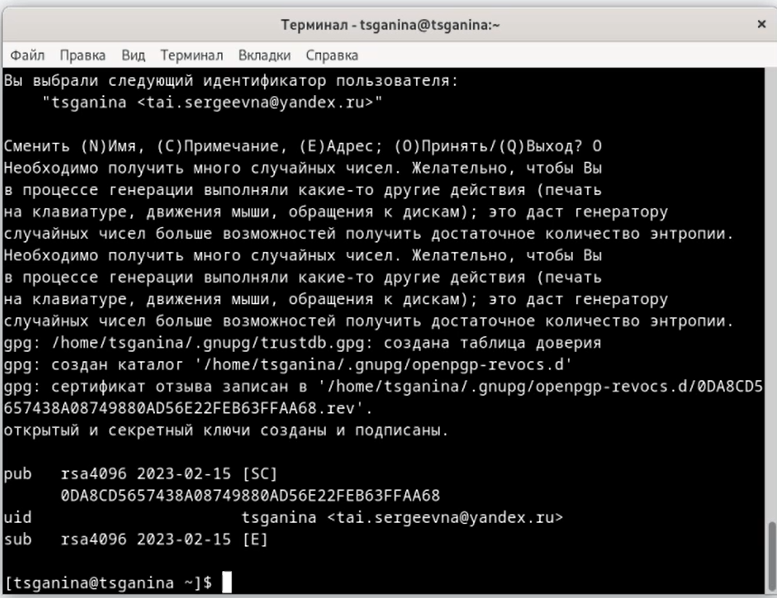


Figure 8: Генерация

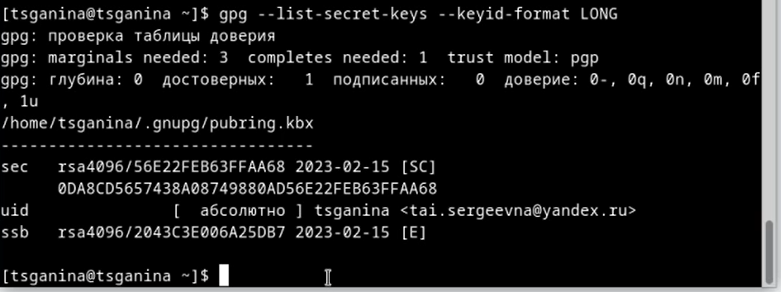


Figure 9: Вывод списка ключей

1. Добавление PGP ключа в GitHub (рис. [10](#fig:010), [11](#fig:011), [12](#fig:012)).

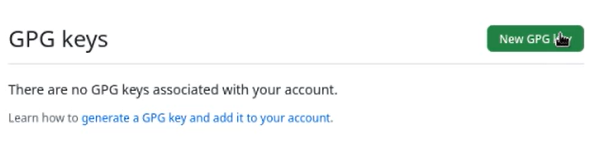


Figure 10: Загрузка ключа

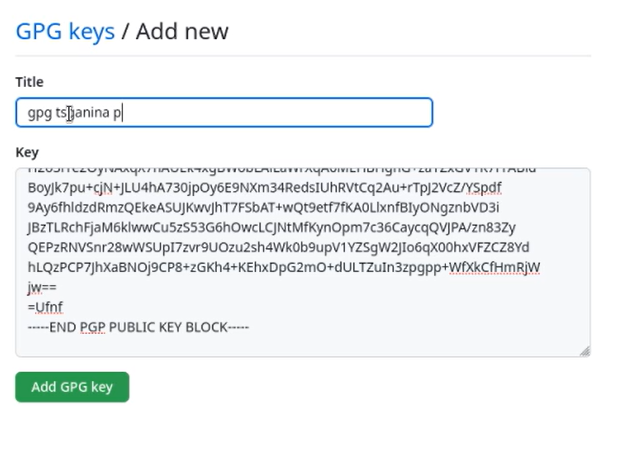


Figure 11: Загрузка ключа

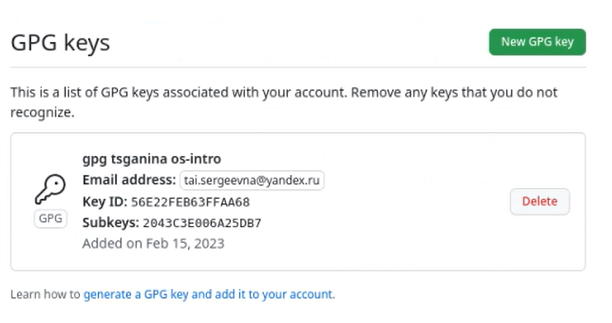


Figure 12: Итог

1. Настройка автоматических подписей коммитов git (рис. [13](#fig:013)).

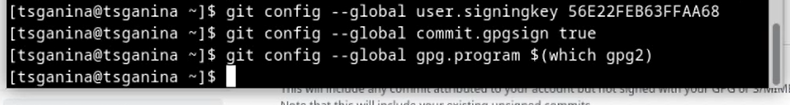


Figure 13: Настройка автоматических подписей коммитов git

1. Настройка gh (рис. [14](#fig:014), [15](#fig:015), [16](#fig:016)).

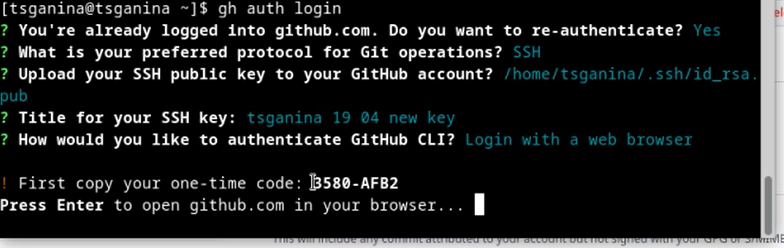


Figure 14: Авторизация

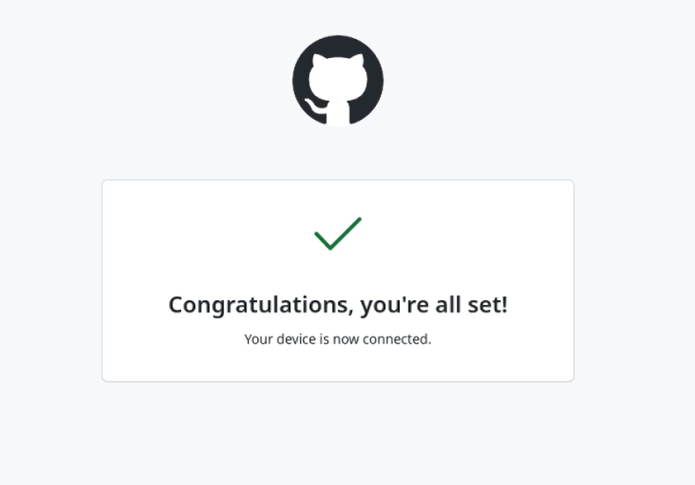


Figure 15: Подключение к браузеру

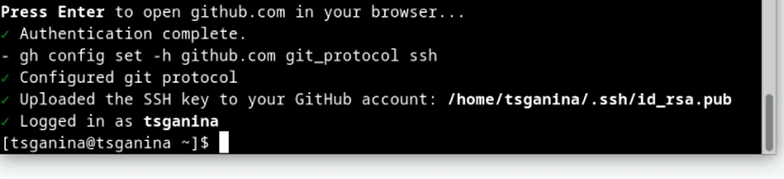


Figure 16: Итог

1. Шаблон для рабочего пространства (рис. [17](#fig:017), [18](#fig:018)).

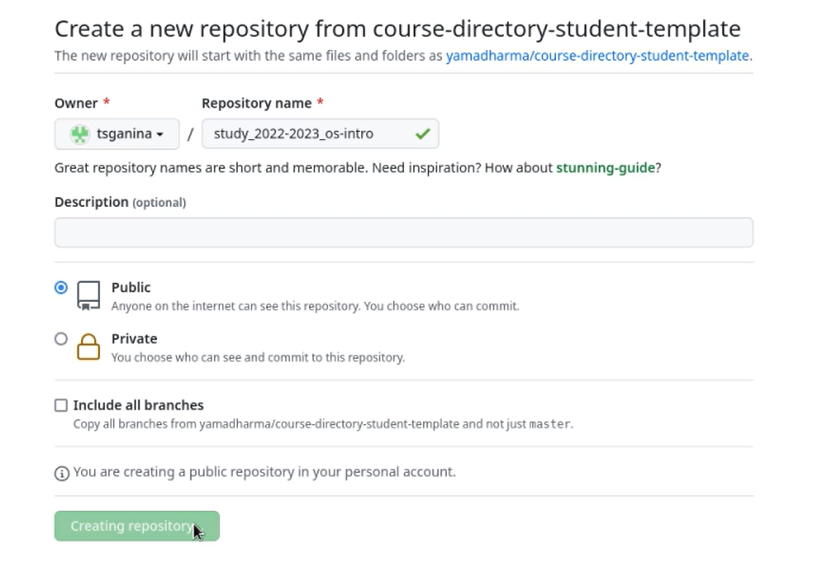


Figure 17: Создание репозитория на основе шаблона

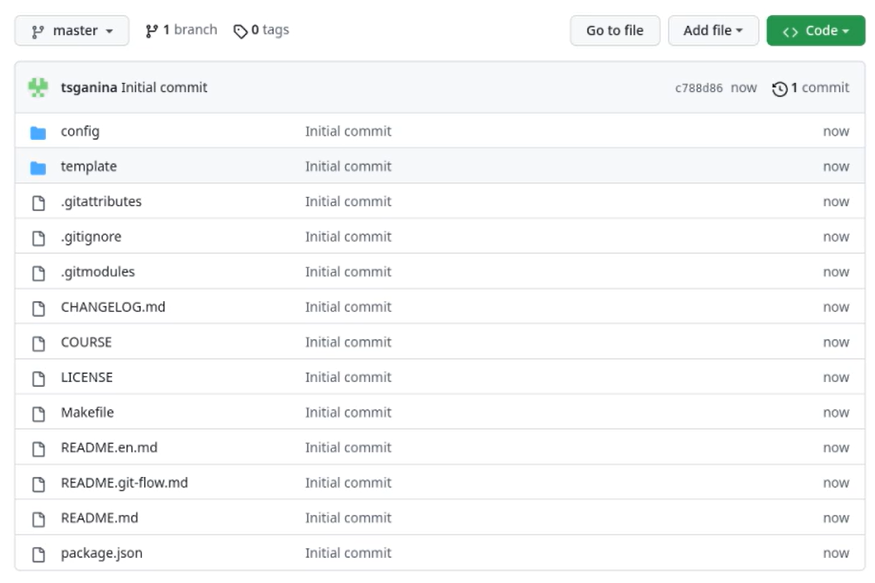


Figure 18: Репозиторий

1. Создание локального репозитория курса на основе шаблона (рис. [19](#fig:019), [20](#fig:020), [21](#fig:021)).

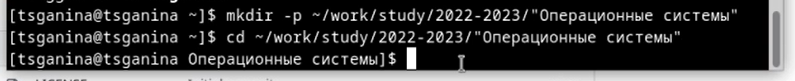


Figure 19: Авторизация

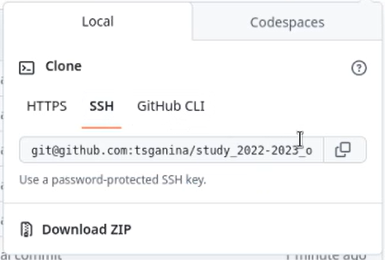


Figure 20: Подключение к браузеру

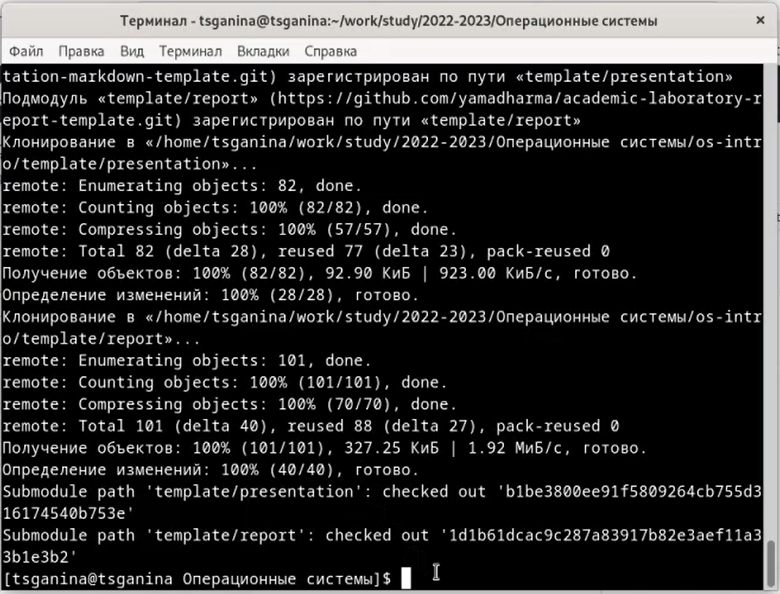


Figure 21: Итог

1. Настройка каталога курса (рис. [22](#fig:022), [23](#fig:023), [24](#fig:024), [25](#fig:025)).

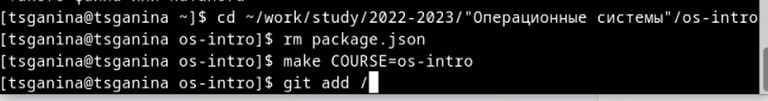


Figure 22: Переход в каталог, удаление лишних файлов, создание каталогов

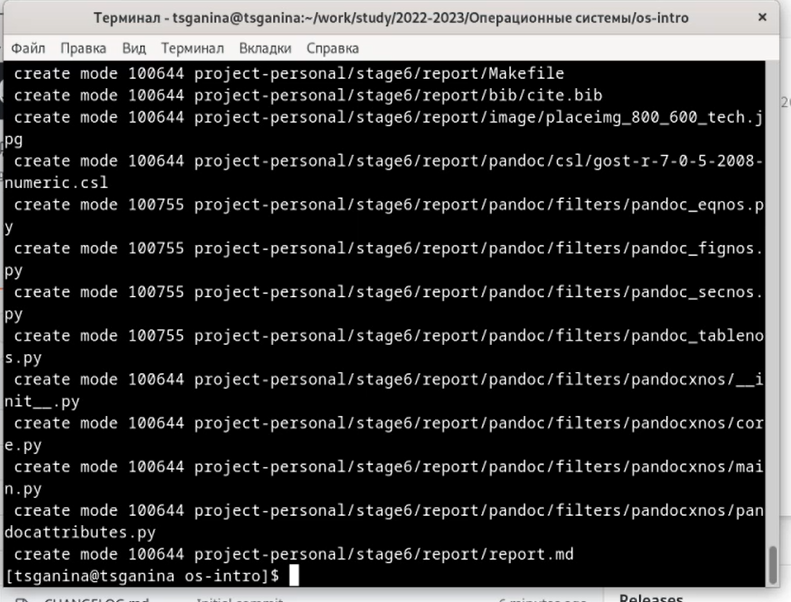


Figure 23: Отправка файлов на сервер

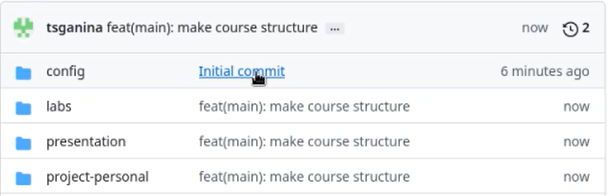


Figure 24: Отправка файлов на сервер

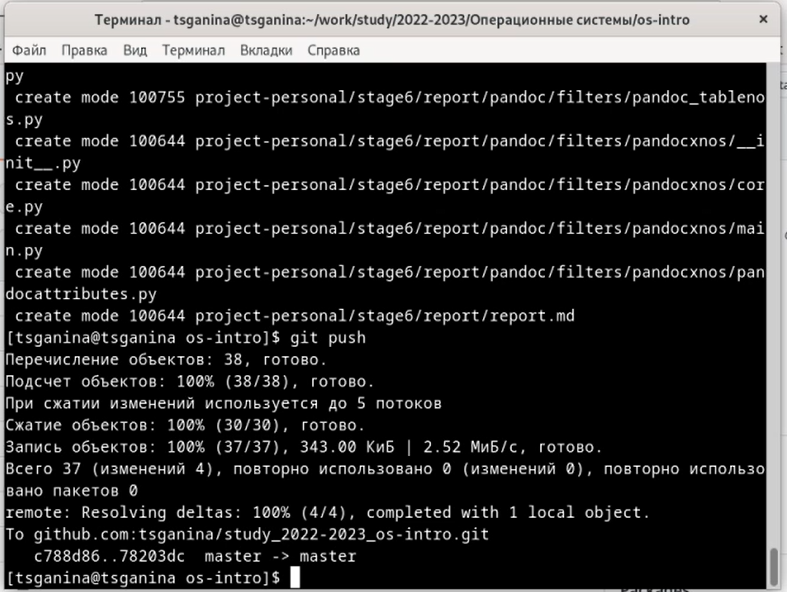


Figure 25: Отправка файлов на сервер

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

Система контроля версий - 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.

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

* Хранилище - репозиторий, хранилище версий, в нем хранятся все документы вместе с их историей, и другой служебной информацией.
* Commit - отслеживание изменений, сохранение разницы в рабочих изменениях.
* История - сохранение всех изменений в проекте и при необходимости возможность обратиться к старым данным.
* Рабочая копия - копия проекта, связанная с репозиторием, текущее состояние файлов проекта, основанное на их последней версии из хранилища.

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

* Централизованные VCS - одно основное хранилище всего проекта, каждый пользователь копирует себе необходимые файлы из репозитория и изменяет их, а потом может добавить обратно. Примеры: - Subversion; - CVS; - TFS; - VAULT; - AccuRev;
* Децентрализованные VCS - У каждого пользователя свой вариант репозитория, есть возможность добавлять и забирать версии из любого репозитория. Например: - Git; - Mercurial; - Bazaar;

Обычно используются централизованные системы контроля версий, с одним общим репозиторием.

1. Опишите действия с VCS при единоличной работе с хранилищем. Сначала нужно создать удалённый репозиторий, после - подключить его. Затем по мере того, как проект будет выполняться, нужно отправлять данные на сервер.
2. Опишите порядок работы с общим хранилищем VCS. В начале при помощи последовательности команд пользователь получает нужную ему версию данных. Потом он работает с ними, вносит некоторые изменения, и уже после этого он может разместить новую версию в хранилище. При этом предыдущие версии не удаляются из хранилища, что является очень удобным - к ним можно вернуться в любой момент.
3. Каковы основные задачи, решаемые инструментальным средством git?

* Хранить информацию обо всех изменениях, производимых в проекте.
* Обеспечить командную работу.

1. 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 изменений

git add . / git add <имя файла> / git rm <имя файла> - Сохранение текущих изменений

git commit / git commit -am "описание коммита" - Сохранение добавленных изменений

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. Приведите примеры использования при работе с локальным и удалённым репозиториями. git push -all (push origin master/любой branch)
2. Что такое и зачем могут быть нужны ветви (branches)? Ветвь - один из параллельных участков истории проекта в одном хранилище. Все ветви исходят из одной версии - точки ветвления. Обычно ветви делятся на master и trunk. Между ветками возможно и слияние. Ветки нужны для разработки новых функций.
3. Как и зачем можно игнорировать некоторые файлы при commit? Игнорировать некоторые файлы можно прописав шаблон .gitignore специально для игнорируемых файлов. Зачем это нужно? Чтобы в репозиторий не попали “лишние” файлы, которые неминуемо будут возникать при работе над проектом. Это могут быть временные файлы, объектные файлы.

# 5 Выводы

* Я изучила идеологию и применение средств контроля версий.
* Освоила умения по работе с git.

# Список литературы

1. [Руководство к выполнению лабораторной работы №2](https://esystem.rudn.ru/mod/page/view.php?id=970819#org853ec93)
2. [Solving “Fatal: Not A Git Repository” (Or Any Of The Parent Directories) Error](https://komodor.com/blog/solving-fatal-not-a-git-repository-error/)