

Отчёт по лабораторной работе №2

Операционные системы

Луангсуваннавонг Сайпхачан

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Установка программного обеспечения	9
4.2	Базовая настройка git	9
4.3	Создание ключа SSH	10
4.4	Создание ключа GPG	11
4.5	Настройка GitHub	13
4.6	Добавление GPG ключа в GitHub	13
4.7	Настройка автоматических подписей коммитов git	15
4.8	Настройка gh	15
4.9	Создание репозитория курса на основе шаблона	16
4.10	Настройка каталога курса	17
5	Выводы	19
6	Ответы на контрольные вопросы	20
	Список литературы	23

Список иллюстраций

4.1	Установка git и gh	9
4.2	Настройка имени и email владельца репозитория	9
4.3	Настройка utf-8 при выводе сообщений git	10
4.4	Установка имени начальной ветви	10
4.5	Настройка параметра autocrlf	10
4.6	Настройка параметра safecrlf	10
4.7	создание ssh-ключа по алгоритму rsa	11
4.8	создание ssh-ключа по алгоритму ed25519	11
4.9	Генерация ключа GPG	12
4.10	Ввод фразу-пароля	12
4.11	Аккаунт на GitHub	13
4.12	Отображение списка ключей	13
4.13	Копирование ключей GPG	14
4.14	Настройка GitHub	14
4.15	Добавление ключа GPG	14
4.16	Добавлен ключ GPG	15
4.17	Настройка подписи GPG	15
4.18	Авторизация в gh	15
4.19	Завершение авторизации через браузер	16
4.20	Завершение авторизации	16
4.21	Создание нового каталога и репозитория	16
4.22	Клонирование репозитория	17
4.23	Перемещение между каталогами	17
4.24	Удаление файлов и создание каталогов	17
4.25	Добавление файлов и комментирование к ним	18
4.26	Отправка файлов на сервер	18

Список таблиц

1 Цель работы

Цель работы - Изучение идеологии и применения инструментов контроля версий и овладение навыками работы с git.

2 Задание

1. Создать базовую конфигурацию для работы с git.
2. Создать ключ SSH.
3. Создать ключ PGP.
4. Настроить подписи git.
5. Зарегистрироваться на Github.
6. Создать локальный каталог для выполнения заданий по предмету.

3 Теоретическое введение

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.

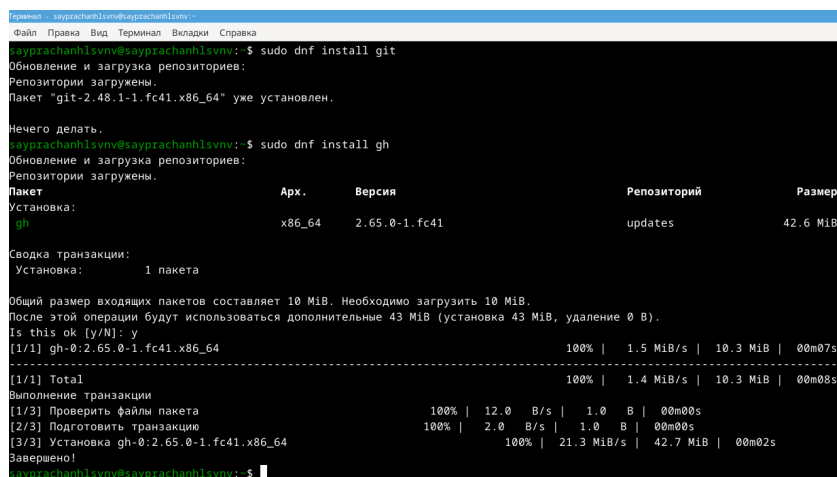
В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным.

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.

4 Выполнение лабораторной работы

4.1 Установка программного обеспечения

Я установил git и gh в систему используя команду `dnf install` (рис. 4.1)



```
sayprachanhlsnv@sayprachanhlsnv:~$ sudo dnf install git
Обновление и загрузка репозитория:
Репозитории загружены.
Пакет "git-2.48.1-1.fc41.x86_64" уже установлен.

Нечего делать.
sayprachanhlsnv@sayprachanhlsnv:~$ sudo dnf install gh
Обновление и загрузка репозитория:
Репозитории загружены.

Пакет
Установка:
    Арх.      Версия      Репозиторий      Размер
    gh       x86_64      2.65.0-1.fc41    updates          42.6 MiB

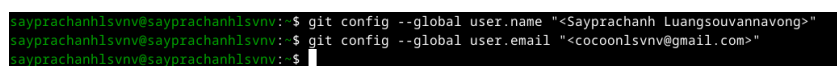
Сводка транзакции:
Установка:      1 пакета

Общий размер входящих пакетов составляет 10 MiB. Необходимо загрузить 10 MiB.
После этой операции будут использоваться дополнительные 43 MiB (установка 43 MiB, удаление 0 B).
Is this ok [y/N]: y
[1/1] gh-0:2.65.0-1.fc41.x86_64 100% | 1.5 MiB/s | 10.3 MiB | 00m07s
-----
[1/1] Total 100% | 1.4 MiB/s | 10.3 MiB | 00m08s
Выполнение транзакции
[1/3] Проверить файлы пакета 100% | 12.0 B/s | 1.0 B | 00m00s
[2/3] Подготовить транзакцию 100% | 2.0 B/s | 1.0 B | 00m00s
[3/3] Установка gh-0:2.65.0-1.fc41.x86_64 100% | 21.3 MiB/s | 42.7 MiB | 00m02s
Завершено!
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.1: Установка git и gh

4.2 Базовая настройка git

Затем я настраиваю имя и адрес электронной почты владельца репозитория, используя свое имя и адрес электронной почты (рис. 4.2)



```
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global user.name "Sayprachanh Luangsouvanavong"
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global user.email "cocoonlsnv@gmail.com"
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.2: Настройка имени и email владельца репозитория

Я настраиваю utf-8 при выводе git-сообщений о коррекции отображения (рис. 4.3)

```
sayprachanhlsvnnv@sayprachanhlsvnnv:~$ git config --global core.quotepath false
sayprachanhlsvnnv@sayprachanhlsvnnv:~$
```

Рис. 4.3: Настройка utf-8 при выводе сообщений git

Я задаю имя начальной ветви как (master) (рис. 4.4)

```
sayprachanhlsvnnv@sayprachanhlsvnnv:~$ git config --global init.defaultBranch master
sayprachanhlsvnnv@sayprachanhlsvnnv:~$
```

Рис. 4.4: Установка имени начальной ветви

Я задаю параметры `autocrlf` и `safecrlf` для корректного отображения строки (рис. 4.5 и рис. 4.6)

```
sayprachanhlsvnnv@sayprachanhlsvnnv:~$ git config --global core.autocrlf input
sayprachanhlsvnnv@sayprachanhlsvnnv:~$
```

Рис. 4.5: Настройка параметра `autocrlf`

```
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global core.safecrlf warn
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.6: Настройка параметра `safercrlf`

4.3 Создание ключа SSH

Я создаю ssh-ключ размером 4096 бит по алгоритму rsa(рис. 4.7)


```
терминал - sayprachanhlsnv@sayprachanhlsnv:~
Файл  Правка  Вид  Терминал  Вкладки  Справка

sayprachanhlsnv@sayprachanhlsnv:~$ gpg --full-generate-key
gpg (GnuPG) 2.4.5; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: создан каталог '/home/sayprachanhlsnv/.gnupg'
Выберите тип ключа:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (только для подписи)
 (14) Existing key from card
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
  0 = не ограничен
  <n> = срок действия ключа - n дней
  <n>w = срок действия ключа - n недель
  <n>m = срок действия ключа - n месяцев
  <n>y = срок действия ключа - n лет
Срок действия ключа? (0) 0
Срок действия ключа не ограничен
Все верно? (y/N) y

GnuPG должен составить идентификатор пользователя для идентификации ключа.

Ваше полное имя: Sayprachanhlsnv
Адрес электронной почты: cocoonlsnv@gmail.com
Примечание:
Вы выбрали следующий идентификатор пользователя:
"Sayprachanhlsnv <cocoonlsnv@gmail.com>"
```

Рис. 4.9: Генерация ключа GPG

Я ввожу фразу-пароль для защиты нового ключа(рис. 4.10)

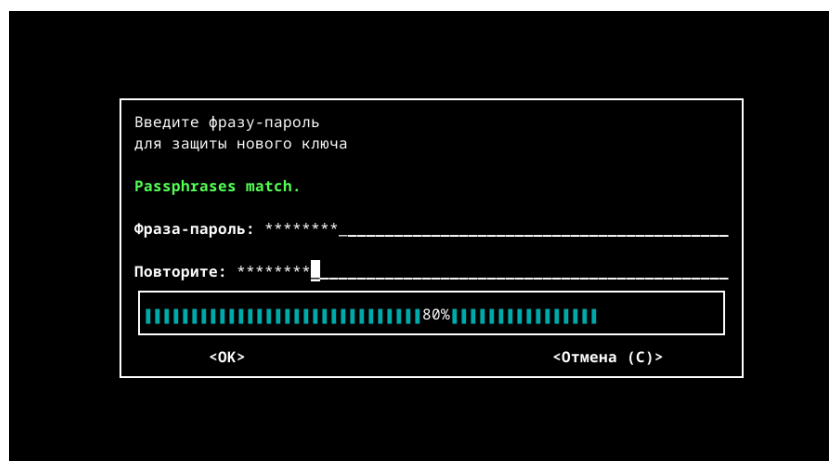


Рис. 4.10: Ввод фразу-пароля

4.5 Настройка GitHub

Я уже создал аккаунт на GitHub, а также настроил систему, поэтому просто вхожу в свой аккаунт.(рис. 4.11)

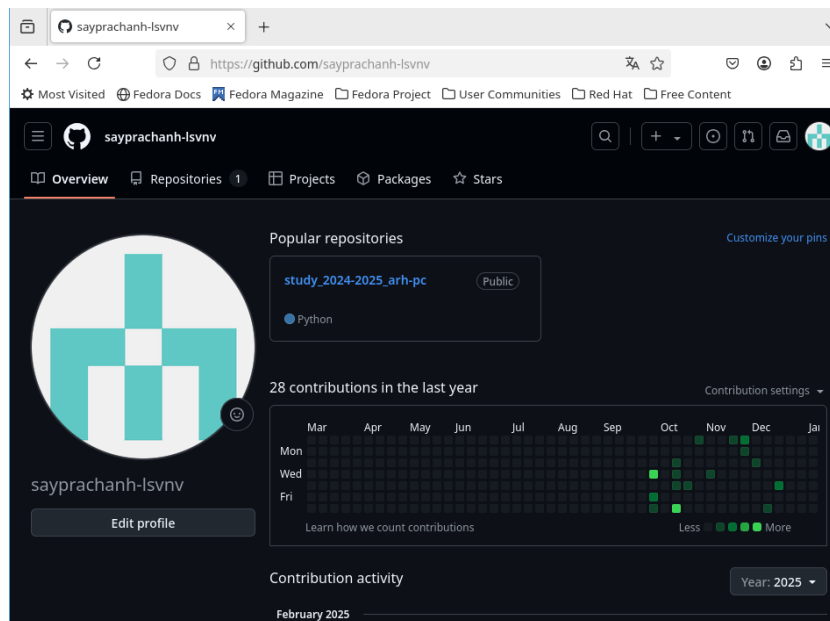


Рис. 4.11: Аккаунт на GitHub

4.6 Добавление GPG ключа в GitHub

Я отображаю список ключей в терминале и нахожу сгенерированные ключи GPG, ключ находится за обратной косой чертой, поэтому я копирую его(рис. 4.12)

```
sayprachanhlsnv@sayprachanhlsnv:~$ gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: глубина: 0 достоверных: 1 подписанных: 0 доверие: 0-, 0q, 0n, 0m, 0f, 1u
[keyboard]
-----
sec   rsa4096/CDDCA4675F3E7037 2025-02-24 [SC]
      ADA3E607351C9D5CC7067E9FCDDCA4675F3E7037
uid           [ абсолютно ] Sayprachanhlsnv <cocoonlsnv@gmail.com>
ssb   rsa4096/C6C708C4DE5935E5 2025-02-24 [E]

sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.12: Отображение списка ключей

Я самостоятельно копирую отпечаток GPG, затем с помощью утилиты xclip

копирую его в буфер обмена(рис. 4.13)

```
sayprachanh1svnv@sayprachanh1svnv: $ gpg --armor --export CDDCA4675F3E7037 | xclip -sel clip
sayprachanh1svnv@sayprachanh1svnv: $
```

Рис. 4.13: Копирование ключей GPG

Захожу в настройки свой аккаунт на GitHub, нахожу раздел GPG keys для ее добавления(рис. 4.14)

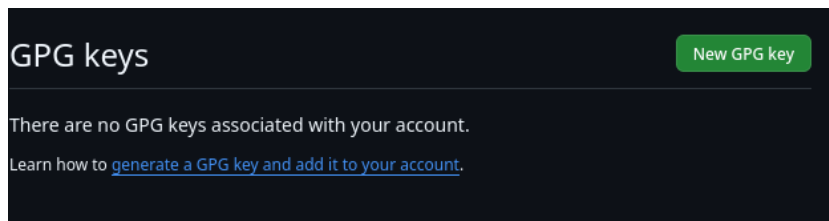


Рис. 4.14: Настройка GitHub

Нажимаю “New GPG Key” и вставляю ключ из буфера обмена(рис. 4.15)

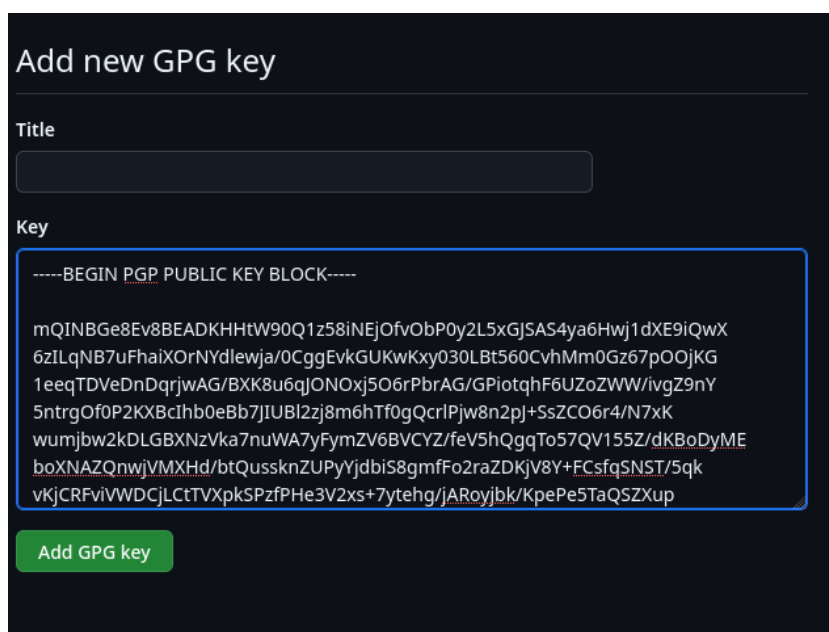


Рис. 4.15: Добавление ключа GPG

Я добавил ключи GPG(рис. 4.16)

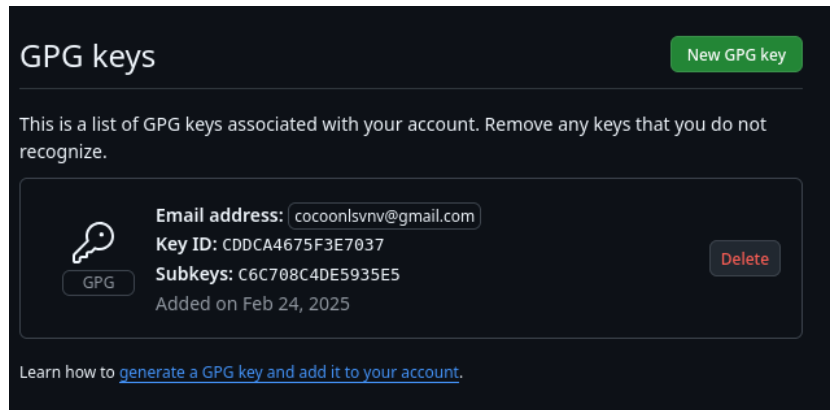


Рис. 4.16: Добавлен ключ GPG

4.7 Настройка автоматических подписей коммитов git

Используя ранее введенный адрес электронной почты, я указываю git, чтобы использовать его при создании подписи фиксации(рис. 4.17)

```
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global user.signingkey CDDCA4675F3E7037
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global commit.gpgsign true
sayprachanhlsnv@sayprachanhlsnv:~$ git config --global gpg.program $(which gpg2)
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.17: Настройка подписи GPG

4.8 Настройка gh

Я начинаю авторизацию в gh, отвечаю на наводящие вопросы и выбираю Авторизоваться через браузер.(рис. 4.18)

```
sayprachanhlsnv@sayprachanhlsnv:~$ gh auth login
? Where do you use GitHub? GitHub.com
? What is your preferred protocol for Git operations on this host? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser
```

Рис. 4.18: Авторизация в gh

Затем я завершаю авторизацию на сайте(рис. 4.19)

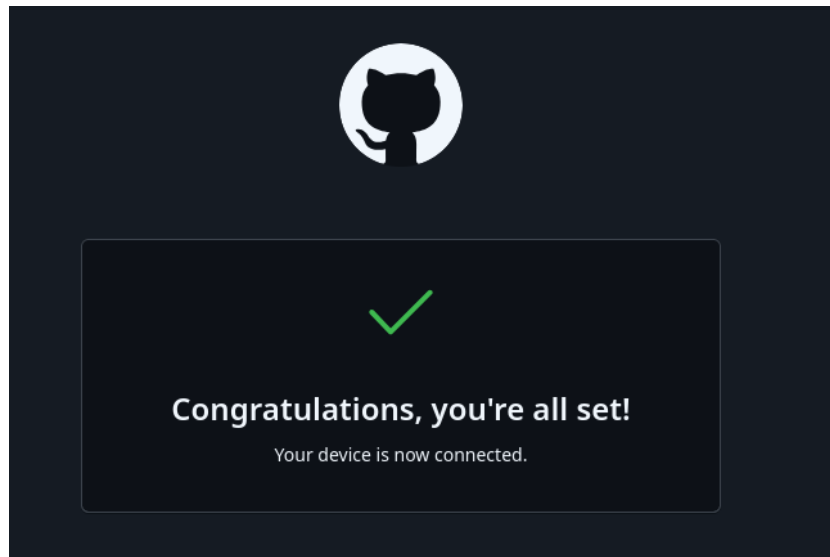


Рис. 4.19: Завершение авторизации через браузер

Виджу текст о завершении авторизации под именем sayprachanh-lsvnv(рис. 4.20)

```
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
! Authentication credentials saved in plain text
✓ Logged in as sayprachanh-lsvnv
sayprachanhlsnv@sayprachanhlsnv:~$
```

Рис. 4.20: Завершение авторизации

4.9 Создание репозитория курса на основе шаблона

После этого я начинаю создавать каталог с помощью mkdir, присваивая ему имя: операционные системы. Используя cd, я захожу в созданный каталог. А также создайте новый репозиторий на моем Github под названием study_2024-2025_os-intro(рис. 4.21)

```
sayprachanhlsnv@sayprachanhlsnv:~$ mkdir -p ~/work/study/2024-2025/Операционные системы
sayprachanhlsnv@sayprachanhlsnv:~$ cd ~/work/study/2024-2025/Операционные системы/
sayprachanhlsnv@sayprachanhlsnv:~/work/study/2024-2025/Операционные системы$ gh repo create study_2024-2025-os-intro --template=yamadharma/course-directory-student-template --public
Created repository sayprachanh-lsvnv/study_2024-2025-os-intro on GitHub
https://github.com/sayprachanh-lsvnv/study_2024-2025-os-intro
sayprachanhlsnv@sayprachanhlsnv:~/work/study/2024-2025/Операционные системы$
```

Рис. 4.21: Создание нового каталога и репозитория

Используя `git clone`, я клонирую удаленный репозиторий, который я создал ранее, в свой локальный репозиторий(рис. 4.22)

```
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы$ git clone --recursive https://github.com/sayprachanhlsnv/study_2024-2025_os-intro.git os-intro
Клонирование в «os-intro»...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 36 (delta 1), reused 21 (delta 0), pack-reused 0 (from 0)
Получение объектов: 100% (36/36), 19.38 КиБ | 211.00 КиБ/с, готово.
Определение изменений: 100% (1/1), готово.
Подмодуль «template/presentation» (https://github.com/yamadharma/academic-presentation-markdown-template.git) зарегистрирован по пути «template/presentation»
Подмодуль «template/report» (https://github.com/yamadharma/academic-laboratory-report-template.git) зарегистрирован по пути «template/report»
Клонирование в «/home/sayprachanhlsnv/work/study/2024-2025/Операционные системы/os-intro/template/presentation»...
```

Рис. 4.22: Клонирование репозитория

4.10 Настройка каталога курса

Я перехожу в каталог с помощью `command cd` и проверяю содержимое каталога с помощью `ls`(рис. 4.23)

```
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы$ cd os-intro/
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$ ls
CHANGELOG.md  config  COURSE  LICENSE  Makefile  package.json  README.en.md  README.git-flow.md  README.md  template
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$
```

Рис. 4.23: Перемещение между каталогами

Используя команду `rm`, я удаляю файл `package.json`. Затем я создаю необходимые каталоги с помощью `make` и выбираю опцию “`prepare`” для создания структуры каталогов(рис. 4.24)

```
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$ rm package.json
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$ echo os-intro > COURSE
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$ make
Usage:
  make <target>

Targets:
  list           List of courses
  prepare        Generate directories structure
  submodule      Update submodules

sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$ make prepare
sayprachanhlsnv@sayprachanhlsnv: ~/work/study/2024-2025/Операционные системы/os-intro$
```

Рис. 4.24: Удаление файлов и создание каталогов

Я добавляю все файлы для отправки на сервер с помощью команды `git add.` и комментирую их с помощью `git commit`(рис. 4.25)

```

sayprachanhlsnvnsayprachanhlsnvns:~/work/study/2024-2025/Операционные системы/os-intro$ git add .
sayprachanhlsnvnsayprachanhlsnvns:~/work/study/2024-2025/Операционные системы/os-intro$ git commit -am 'feat(main): make course structure'
[master 9de9356] feat(main): make course structure
405 files changed, 98413 insertions(+), 14 deletions(-)
create mode 100644 labs/README.md
create mode 100644 labs/README.ru.md

```

Рис. 4.25: Добавление файлов и комментирование к ним

Я отправляю файл на сервер с помощью команды `git push`(рис. 4.26)

```

sayprachanhlsnvnsayprachanhlsnvns:~/work/study/2024-2025/Операционные системы/os-intro$ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 5 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 342.32 КиБ | 1.20 МБ/с, готово.
Total 38 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/sayprachanhlsnvns/study_2024-2025_os-intro.git
3630cab..9de9356 master -> master
sayprachanhlsnvnsayprachanhlsnvns:~/work/study/2024-2025/Операционные системы/os-intro$

```

Рис. 4.26: Отправка файлов на сервер

5 Выводы

Во время выполнения этой лабораторной работы, я изучил идеологию и применяемые инструменты контроля версий, а также овладел навыками работы с git

6 Ответы на контрольные вопросы

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

Системы контроля версий (VCS) — это инструменты для управления изменениями в файлах. Они помогают:

Сохранять историю изменений. Отслеживать, кто и когда внёс изменения. Возвращаться к предыдущим версиям. Совместно работать над проектами.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия. Хранилище (репозиторий) — место, где хранятся все версии файлов.

Commit — фиксация изменений в хранилище с комментарием. История — последовательность всех commit'ов. Рабочая копия — текущая версия файлов на компьютере разработчика.

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

Централизованные VCS (например, SVN): одно главное хранилище, с которым работают все. Децентрализованные VCS (например, Git): у каждого разработчика своя копия хранилища, изменения объединяются позже.

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

Создать репозиторий (`git init`). Добавить файлы (`git add`). Фиксировать изменения (`git commit`). Просматривать историю (`git log`).

5. Опишите порядок работы с общим хранилищем VCS.

Клонировать репозиторий (`git clone`). Создать ветку для изменений (`git branch`). Фиксировать изменения (`git commit`). Отправить изменения на сервер (`git push`). Получить изменения от других (`git pull`).

6. Каковы основные задачи, решаемые инструментальным средством `git`?

Управление версиями файлов. Совместная работа над кодом. Создание и управление ветками. Резервное копирование и восстановление.

7. Назовите и дайте краткую характеристику командам `git`.

`git init` — создать новый репозиторий. `git add` — добавить файлы в индекс. `git commit` — зафиксировать изменения. `git push` — отправить изменения на сервер. `git pull` — получить изменения с сервера. `git branch` — управление ветками. `git merge` — объединить ветки.

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

Чтобы работать с локальным репозиторием, сначала создаём его с помощью команды `git init`. Затем добавляем файлы командой `git add` и фиксируем изменения командой `git commit -m "Описание изменений"`. Для работы с удалённым репозиторием клонируем его командой `git clone`, вносим изменения и отправляем их на сервер командой `git push origin main`.

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

Ветви — это отдельные линии разработки. Они нужны для: Параллельной работы над разными задачами. Изоляции экспериментальных изменений. Упрощения слияния изменений.

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

Во время работы над проектом могут быть созданы файлы, которые не следует добавлять в репозиторий, например, временный файл. Чтобы игнорировать это, мы создаем файл `.gitignore` и указываем в нем шаблоны файлов или папок.

Список литературы

Лабораторная работа № 2