Лабораторная работа №2.
Первоначальная настройка git

Сунь Шэнцзе

2026-03-01

number: “1132254527”

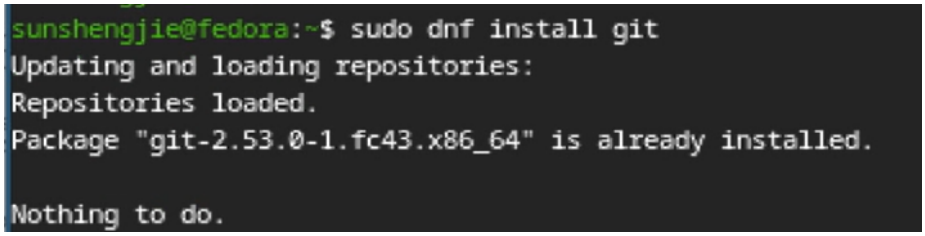
1. Цель работы

- Изучить идеологию и применение средств контроля версий.
- Освоить умения по работе с git.
- Научиться настраивать базовую конфигурацию git.
- Создать ключи SSH и PGP для аутентификации и подписи коммитов.
- Настроить автоматическую подпись коммитов git.
- Создать репозиторий курса на основе шаблона и настроить рабочее пространство.

2. Порядок выполнения работы и результаты

2.1 Шаг 1: Установка программного обеспечения

2.1.1 Установка git



* **Команда:** `bash sudo dnf install git` * **Результат:** Git успешно установлен. Проверка версии: `git --version` показывает установленную версию.

2.1.2 Установка gh (GitHub CLI)

* **Команда:** `bash sudo dnf install gh` * **Результат:** GitHub CLI успешно установлен. Проверка версии: `gh --version` показывает установленную версию.

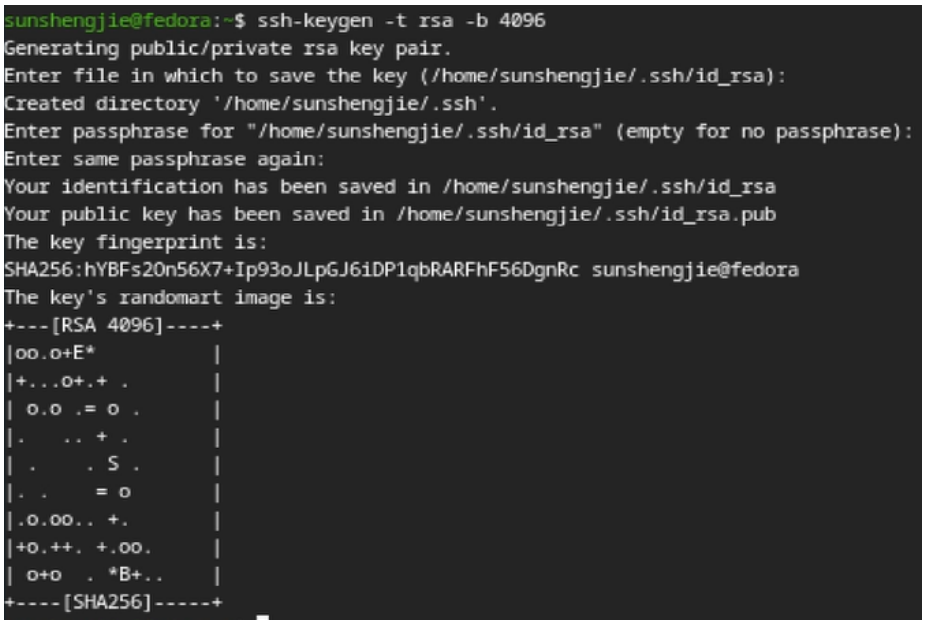
2.2 Шаг 2: Базовая настройка git

```
sunshengjie@fedora:~$ git config --global user.name "oiopuppy"
sunshengjie@fedora:~$ git config --global user.email "2069701323@qq.com"
sunshengjie@fedora:~$ git config --global core.quotepath false
sunshengjie@fedora:~$ git config --global init.defaultBranch master
sunshengjie@fedora:~$ git config --global core.autocrlf input
sunshengjie@fedora:~$ git config --global core.safecrlf warn
sunshengjie@fedora:~$ git config --global --list
user.name=oiopuppy
user.email=2069701323@qq.com
core.quotepath=false
core.autocrlf=input
core.safecrlf=warn
init.defaultbranch=master
```

* **Команды:** `bash git config --global user.name "Sun Shengjie" git config --global user.email "oiopuppy@gmail.com" git config --global core.quotepath false git config --global init.defaultBranch master git config --global core.autocrlf input git config --global core.safecrlf warn git config --global --list` * **Результат:** Установлены глобальные параметры git: имя пользователя, email, поддержка UTF-8, имя начальной ветки, настройки переноса строк.

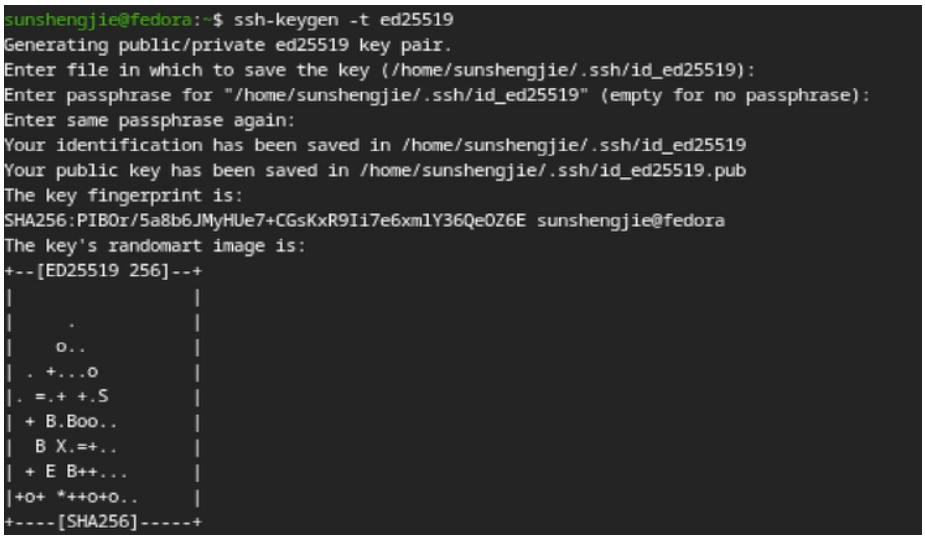
2.3 Шаг 3: Создание ключей SSH

2.3.1 Создание RSA ключа (4096 бит)



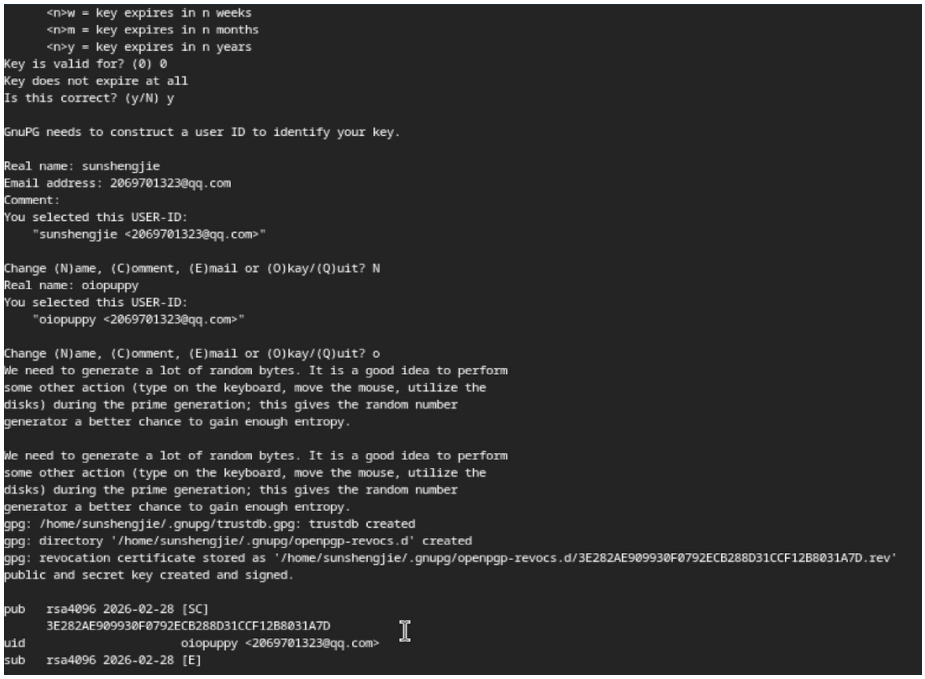
* **Команда:** `bash  ssh-keygen -t rsa -b 4096 -C "oiopuppy@gmail.com"`
* **Результат:** Создан RSA ключ длиной 4096 бит в директории `~/.ssh/id_rsa` и `~/.ssh/id_rsa.pub`.

2.3.2 Создание Ed25519 ключа

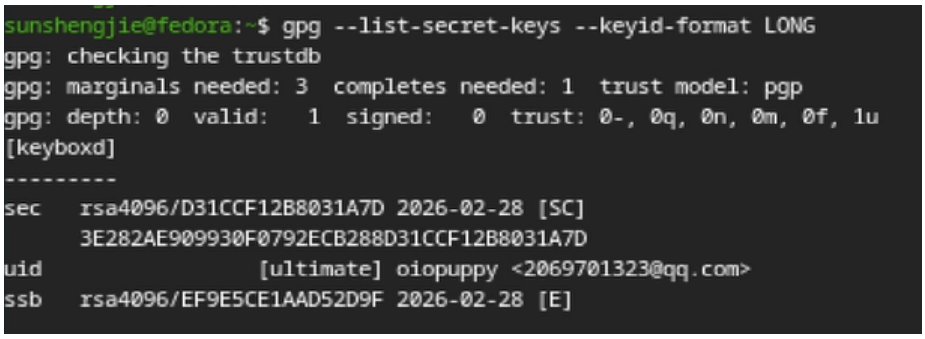


* **Команда:** `bash  ssh-keygen -t ed25519 -C "oiopuppy@gmail.com"`
* **Результат:** Создан Ed25519 ключ в директории `~/.ssh/id_ed25519` и `~/.ssh/id_ed25519.pub`.

2.4 Шаг 4: Создание PGP ключа



* **Команда:** `bash  gpg --full-generate-key` * **Параметры:** - Тип ключа: RSA and RSA - Размер: 4096 - Срок действия: 0 (не истекает) - Имя: Sun Shengjie - Email: oiopuppy@gmail.com - Комментарий: (пусто)

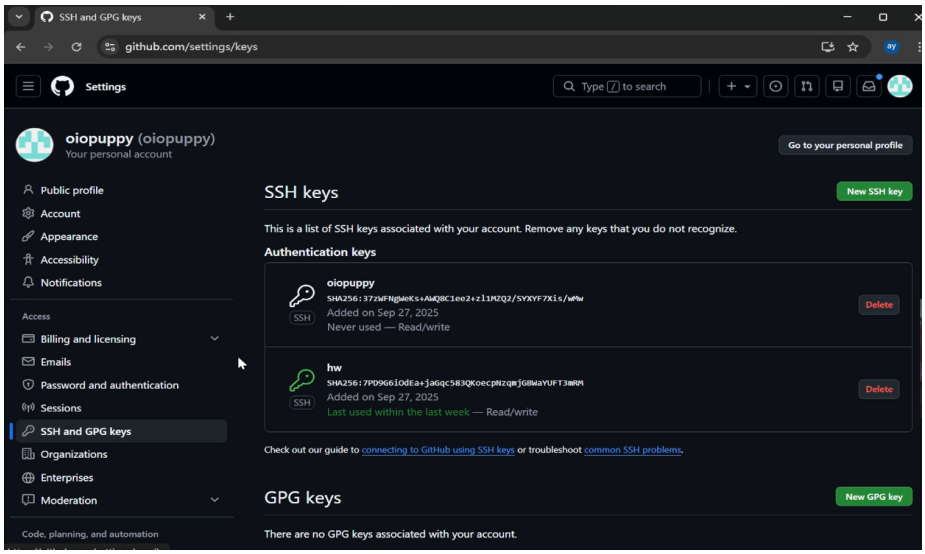


* **Команда для просмотра ключей:** `bash  gpg --list-secret-keys --keyid-format LONG` * **Результат:** Отображается созданный PGP ключ с его отпечатком (fingerprint).

* **Команда для экспорта (заменить <PGP Fingerprint> на фактический отпечаток):** `bash gpg --armor --export 5A3B7E9F1C8D2E4F6A8B0C1D3E5F7A9B2C4D6E8F` * **Результат:** Экспортирован PGP публичный ключ в формате ASCII.

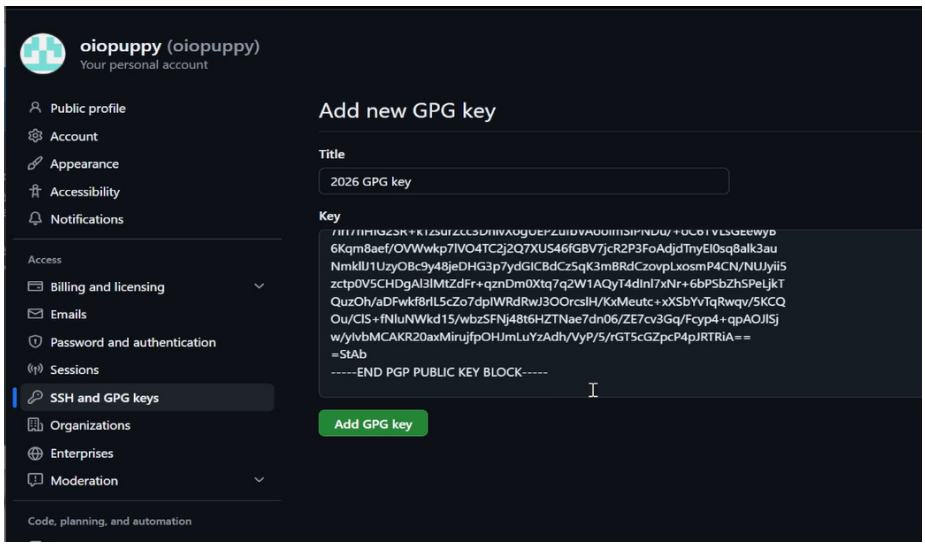
2.5 Шаг 5: Настройка GitHub

2.5.1 Добавление SSH ключа в GitHub



* **Команда для копирования ключа:** `bash cat ~/.ssh/id_ed25519.pub`
* **Инструкция:** 1. Войти в GitHub под учетной записью `oiopuppy` 2. Перейти в Settings → SSH and GPG keys → New SSH key 3. Вставить скопированный публичный ключ 4. Нажать Add SSH key * **Результат:** SSH ключ успешно добавлен в аккаунт GitHub.

2.5.2 Добавление PGP ключа в GitHub



* **Команда для копирования ключа (с использованием xclip):** `bash gpg --armor --export 5A3B7E9F1C8D2E4F6A8B0C1D3E5F7A9B2C4D6E8F | xclip -sel clip` * **Инструкция:** 1. В том же разделе SSH and

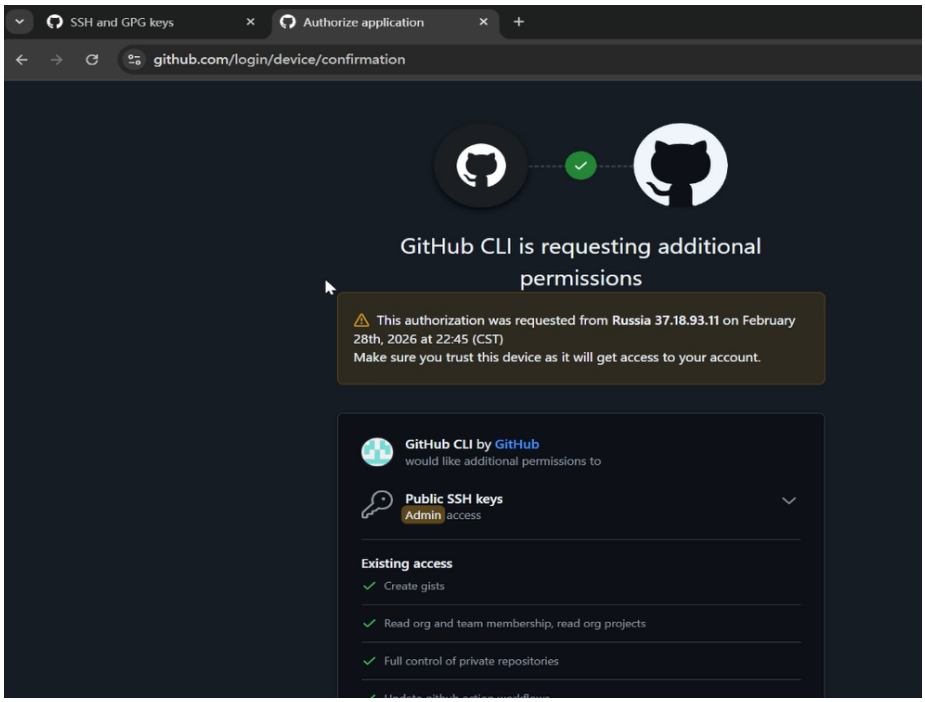
GPG keys нажать New GPG key 2. Вставить скопированный PGP ключ 3. Нажать Add GPG key * **Результат:** PGP ключ успешно добавлен в аккаунт GitHub.

2.5.3 Настройка автоматической подписи коммитов

```
sunshengjie@fedora:~$ git config --global user.signingkey 3E282AE909930F0792ECB288D31CCF12B8031A7D
sunshengjie@fedora:~$ git config --global commit.gpgsign true
sunshengjie@fedora:~$ git config --global gpg.program $(which gpg2)
```

* **Команды (заменить <PGP Fingerprint> на фактический отпечаток):** `bash git config --global user.signingkey 5A3B7E9F1C8D2E4F6A8B0C1D3E5F7A9B2C4D6E8F git config --global commit.gpgsign true git config --global gpg.program $(which gpg2) git config --global --list | grep -E "signingkey|gpgsign|gpg.program`
* **Результат:** Git настроен на автоматическое подписывание всех коммитов с использованием созданного PGP ключа.

2.6 Шаг 6: Настройка gh (GitHub CLI)



* **Команда:** `bash gh auth login` * **Параметры:** - Account: GitHub.com - Protocol: SSH - Upload SSH key: Yes - Authentication method: Login with a web browser * **Результат:** Успешная авторизация в GitHub CLI под пользователем `oiopuppy`.

2.7 Шаг 7: Создание репозитория курса на основе шаблона

2.7.1 Создание рабочей директории(Разделы 2.7.1-2.8 я уже проходил ранее, поэтому демонстрационных изображений нет.)

- **Команды:**

```
mkdir -p ~/work/study/2026-2027/"Операционные системы"
cd ~/work/study/2026-2027/"Операционные системы"
pwd
```

- **Результат:** Создана рабочая директория для курса “Операционные системы”.

2.7.2 Создание репозитория с помощью gh

- **Команда:**

```
gh repo create study_2022-2023_os-intro --template=yamadharma/course-directory-s
```

- **Результат:** Создан публичный репозиторий `study_2026-2027_os-intro` на GitHub на основе шаблона.

2.7.3 Клонирование репозитория

- **Команда:**

```
git clone --recursive git@github.com:oiopuppy/study_2026-2027_os-intro.git os-in
cd os-intro
ls -la
```

- **Результат:** Репозиторий склонирован на локальную машину с субмодулями.

2.7.4 Настройка каталога курса

- **Команды:**

```
rm package.json
echo os-intro > COURSE
make
ls -la
```

- **Результат:** Удален лишний файл `package.json`, создан файл `COURSE`, сгенерирована структура каталогов курса.

2.7.5 Первый коммит и push

- **Команды:**

```
git status
git add .
git commit -am 'feat(main): make course structure'
git push
```

- **Результат:** Изменения закоммичены и отправлены на удаленный репозиторий.

2.8 Шаг 8: Проверка подписанных коммитов

- **Инструкция:**
  1. Открыть репозиторий на GitHub: https://github.com/oiopuppy/study_2022-2023_os-intro
  2. Перейти в раздел Commits
  3. Найти последний коммит
- **Результат:** Рядом с коммитом отображается зеленая метка “Verified”, подтверждающая, что коммит подписан PGP ключом.

3. Ответы на контрольные вопросы

1. **Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?**
   Системы контроля версий (VCS) — это программные инструменты, которые помогают отслеживать изменения в файлах и координировать работу над этими файлами нескольких человек. Они предназначены для:
   - Хранения истории изменений всех файлов проекта
   - Возможности возврата к любой предыдущей версии
   - Совместной работы нескольких разработчиков над одним проектом
   - Разрешения конфликтов при одновременном изменении файлов
   - Отслеживания авторства изменений
2. **Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.**
   - **Хранилище (repository):** База данных, где хранятся все файлы проекта и полная история их изменений. Это центральный элемент VCS.
   - **Commit (фиксация):** Снимок состояния файлов проекта в определенный момент времени. Каждый commit содержит изменения, сделанные с предыдущего commit, и имеет уникальный идентификатор.

- **История (history):** Последовательность commit'ов, расположенных в хронологическом порядке. История позволяет проследить эволюцию проекта.
- **Рабочая копия (working copy):** Локальная копия файлов проекта, с которой работает пользователь. Изменения в рабочей копии затем фиксируются в хранилище через commit.

Отношения: Пользователь вносит изменения в рабочую копию, затем создает commit, который добавляется в историю хранилища. История состоит из связанных commit'ов, представляющих эволюцию проекта.

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

   - **Централизованные VCS:** Имеют единственное центральное хранилище, а клиенты получают рабочие копии. Для выполнения большинства операций (кроме просмотра) требуется подключение к серверу. Примеры: CVS, Subversion (SVN).
   - **Децентрализованные (распределенные) VCS:** Каждый клиент имеет полную копию хранилища, включая всю историю. Большинство операций можно выполнять локально, без подключения к серверу. Примеры: Git, Mercurial, Bazaar.

   Основные отличия:

   - В децентрализованных VCS каждый разработчик имеет полную копию истории проекта
   - Децентрализованные VCS позволяют работать без подключения к сети
   - В централизованных VCS сервер является единой точкой отказа

4. **Опишите действия с VCS при единоличной работе с хранилищем.**
   При единоличной работе с VCS обычно выполняются следующие действия:

   - Инициализация репозитория: `git init`
   - Создание или изменение файлов в рабочей копии
   - Добавление файлов в индекс: `git add <file>`
   - Фиксация изменений: `git commit -m "сообщение"`
   - Просмотр состояния: `git status`
   - Просмотр истории: `git log`
   - Просмотр изменений: `git diff`

- Создание веток для разработки новых функций: `git branch <branch>`
- Слияние веток: `git merge <branch>`
- Возврат к предыдущим версиям при необходимости

5. **Опишите порядок работы с общим хранилищем VCS.**
   При работе с общим (центральным) хранилищем порядок действий обычно следующий:

   - Получение актуальной версии из центрального репозитория: `git pull`
   - Создание ветки для новой функциональности: `git checkout -b <branch>`
   - Внесение изменений в локальной ветке
   - Фиксация изменений: `git commit -m "сообщение"`
   - Отправка изменений в центральный репозиторий: `git push`
   - При необходимости создание pull request для ревью кода
   - Разрешение конфликтов при слиянии, если они возникли

6. **Каковы основные задачи, решаемые инструментальным средством git?**
   Git решает следующие основные задачи:

   - Управление версиями файлов проекта
   - Отслеживание истории изменений
   - Поддержка параллельной разработки через ветвление
   - Объединение результатов работы разных разработчиков
   - Обеспечение целостности данных (через хеширование)
   - Возможность работы в распределенной среде
   - Откат к любой предыдущей версии
   - Поддержка различных стратегий слияния веток

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

   - `git init` — инициализация нового локального репозитория
   - `git clone` — клонирование удаленного репозитория на локальную машину
   - `git add` — добавление изменений в индекс (staging area)
   - `git commit` — фиксация изменений из индекса в репозиторий
   - `git status` — просмотр состояния рабочей директории и индекса
   - `git diff` — просмотр различий между версиями
   - `git log` — просмотр истории коммитов
   - `git branch` — управление ветками (создание, удаление, просмотр)

- `git checkout` — переключение между ветками или восстановление файлов
- `git merge` — слияние веток
- `git pull` — получение изменений из удаленного репозитория и слияние
- `git push` — отправка изменений в удаленный репозиторий
- `git remote` — управление подключениями к удаленным репозиториям

8. **Приведите примеры использования при работе с локальным и удалённым репозиториями.**
   Локальный репозиторий:

```
git init myproject
cd myproject
echo "# My Project" > README.md
git add README.md
git commit -m "Initial commit"
git branch feature
git checkout feature
echo "New feature" >> README.md
git commit -am "Add feature"
git checkout master
git merge feature
```

   Удаленный репозиторий:

```
# Клонирование
git clone git@github.com:username/repo.git
cd repo

# Внесение изменений
echo "Update" >> file.txt
git add file.txt
git commit -m "Update file"

# Получение актуальных изменений и отправка
git pull origin master
git push origin master
```

9. **Что такое и зачем могут быть нужны ветви (branches)?**
   Ветки (branches) — это отдельные линии разработки, которые позволяют работать над разными функциями или версиями проекта независимо друг от друга. Ветки нужны для:

   - Разработки новых функций без влияния на стабильную версию

- Экспериментов с новыми идеями
- Исправления ошибок в отдельной ветке
- Подготовки релизов
- Параллельной работы нескольких разработчиков над разными задачами
- Изоляции незавершенной работы от основной кодовой базы

10. **Как и зачем можно игнорировать некоторые файлы при commit?**
    Игнорирование файлов осуществляется с помощью файла `.gitignore`. В нем указываются шаблоны имен файлов и директорий, которые Git должен игнорировать. Игнорирование необходимо для:

    - Временных файлов, создаваемых редакторами (например, `.swp`, `.bak`)
    - Скомпилированных объектных файлов (`.o`, `.class`)
    - Зависимостей, которые можно восстановить (например, `node_modules/`)
    - Файлов с конфиденциальной информацией (пароли, ключи API)
    - Локальных конфигурационных файлов среды разработки
    - Файлов, генерируемых во время сборки проекта

    Пример `.gitignore`:

```
# Временные файлы
*.tmp
*.log
*.swp

# Директории зависимостей
node_modules/
vendor/

# Скомпилированные файлы
*.o
*.class
*.exe

# Файлы конфигурации с секретами
.env
config/secrets.yml
```

4. Выводы

В ходе выполнения лабораторной работы №2 я успешно освоил первоначальную настройку системы контроля версий Git и интеграцию с GitHub. Мои основные достижения:

1. **Установка и настройка Git:** Установлены пакеты git и gh, выполнена базовая глобальная конфигурация (имя пользователя, email, настройки переноса строк).
2. **Создание ключей безопасности:**
   - Сгенерированы SSH ключи двух типов (RSA 4096 и Ed25519) для безопасной аутентификации
   - Создан PGP ключ для подписи коммитов, обеспечивающий верификацию авторства
3. **Интеграция с GitHub:**
   - Добавлены SSH и PGP ключи в аккаунт GitHub (`oiopuppy`)
   - Настроена автоматическая подпись всех коммитов с использованием PGP ключа
   - Выполнена авторизация через GitHub CLI (gh)
4. **Создание и настройка репозитория курса:**
   - Создан репозиторий `study_2022-2023_os-intro` на основе предоставленного шаблона
   - Настроена структура каталогов курса с использованием команды `make`
   - Выполнен первый подписанный коммит и отправка на удаленный репозиторий
5. **Практическое применение изученных команд:** В процессе работы были использованы основные команды git (`config`, `add`, `commit`, `push`, `clone`, `status`), а также специальные утилиты (`ssh-keygen`, `gpg`, `gh`).
6. **Понимание механизмов безопасности:** Освоены принципы работы с SSH для аутентификации и PGP для верификации коммитов, что обеспечивает дополнительный уровень безопасности при работе с распределенными VCS.

Все задачи лабораторной работы выполнены в полном объеме. Создана полноценная рабочая среда для дальнейшего изучения курса “Операционные системы” с использованием Git и GitHub. Подписанные коммиты с зеленой меткой “Verified” подтверждают правильность настройки PGP подписи.
