Лабораторная работа 2

Первоначальна настройка git

Иванова Ангелина Олеговна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Изучить идеологию и применение средств контроля версий, а также освоить базовые умения по работе с git.

2 Задание

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

3 Выполнение лабораторной работы

3.1 Установка програмного обеспечения

Установили git (рис. 3.1).

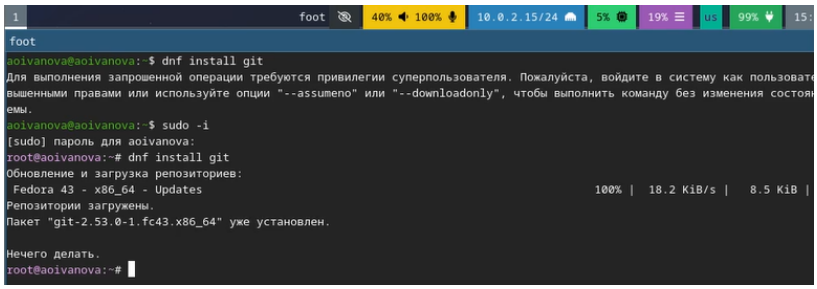

Рисунок 3.1: Установка git

Установили gh (рис. 3.2).

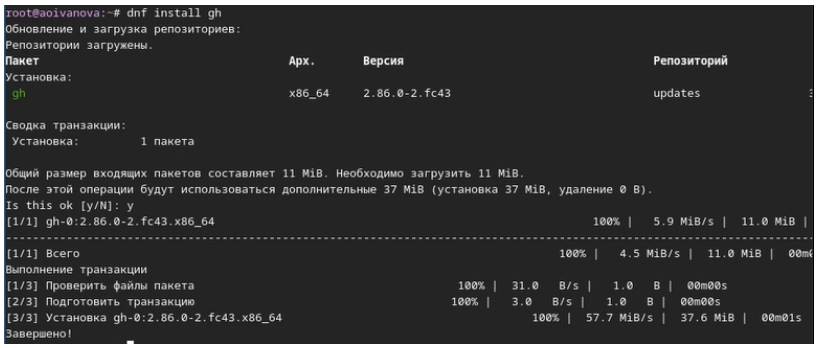

Рисунок 3.2: Установка gh

3.2 Базовая настройка git

Задали имя и email владельца репозитория (рис. 3.3).

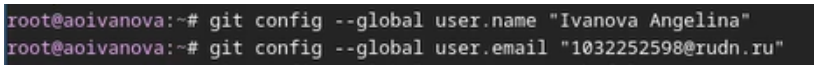

Рисунок 3.3: Установка имени и почты

Настроили utf-8 в выводе сообщений git (рис. 3.4).

Рисунок 3.4: Настройка utf-8

Настроили верификацию и подписание коммитов git. Для этого сгенерировали ключ(рис. 3.5).

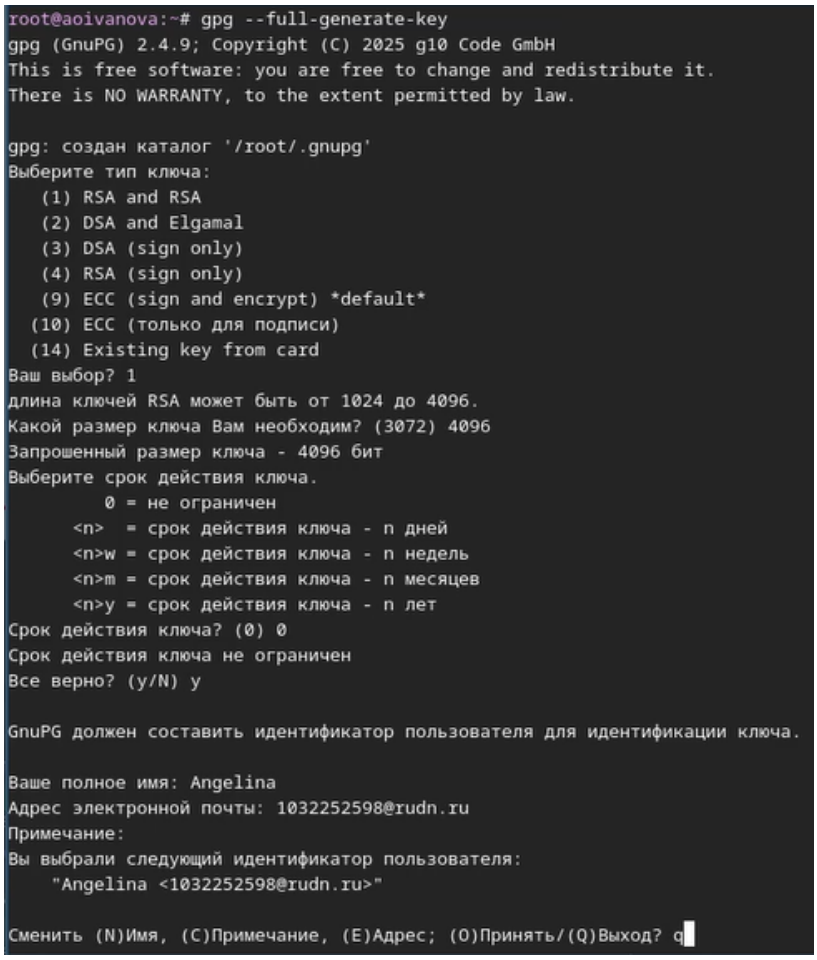

Рисунок 3.5: Генерация ключа

Из предложенных опций выбрали:

- тип RSA and RSA;
- размер 4096;
- срок действия не истекает никогда;
- Имя (Angelina);
- Адрес электронной почты (1032252598@rudn.ru).

Вывели список ключей и скопировали отпечаток приватного ключа(в моем случае BASE624972FBE7B)(рис. 3.6).

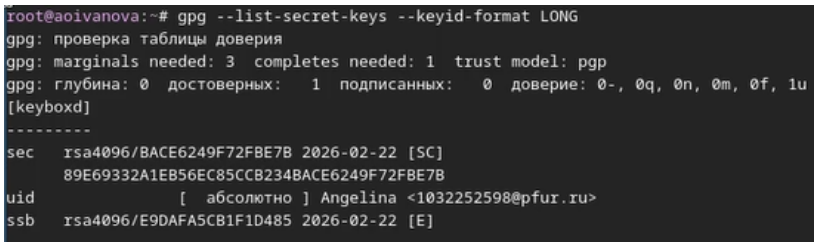

Рисунок 3.6: Вывод списка ключей

Вывели ключ в формате ASCII по его отпечатку и скопировали его(рис. 3.7).

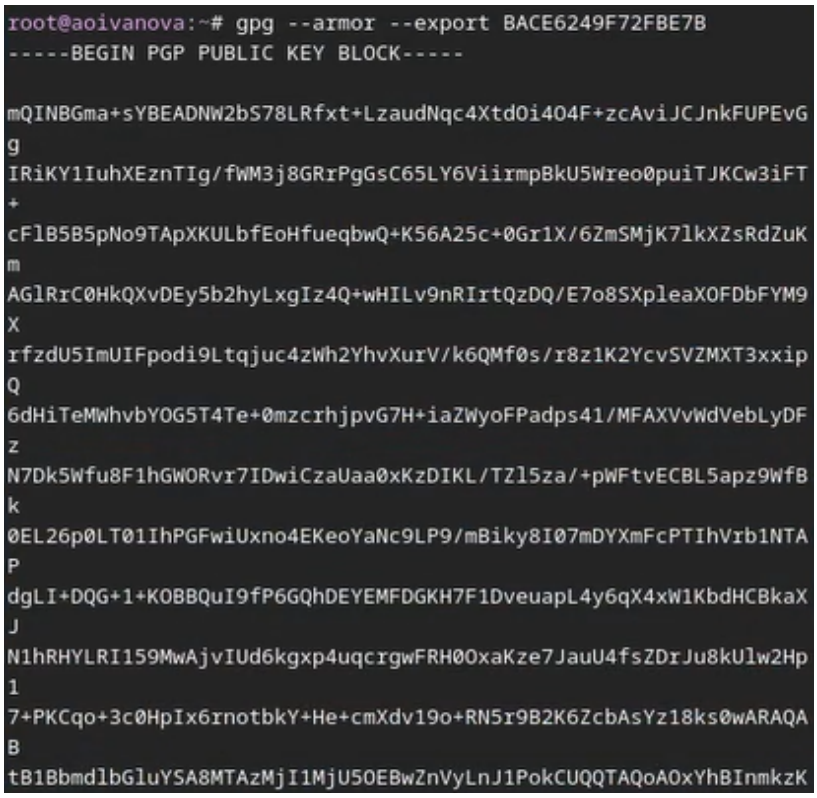

Рисунок 3.7: Вывод списка ключей

Перейkb в настройки GitHub (https://github.com/settings/keys), нажfkb на кнопку New GPG key и вставили полученный ключ в поле ввода, после чего получили готовый загруженный ключ(рис. 3.8),(рис. 3.9).

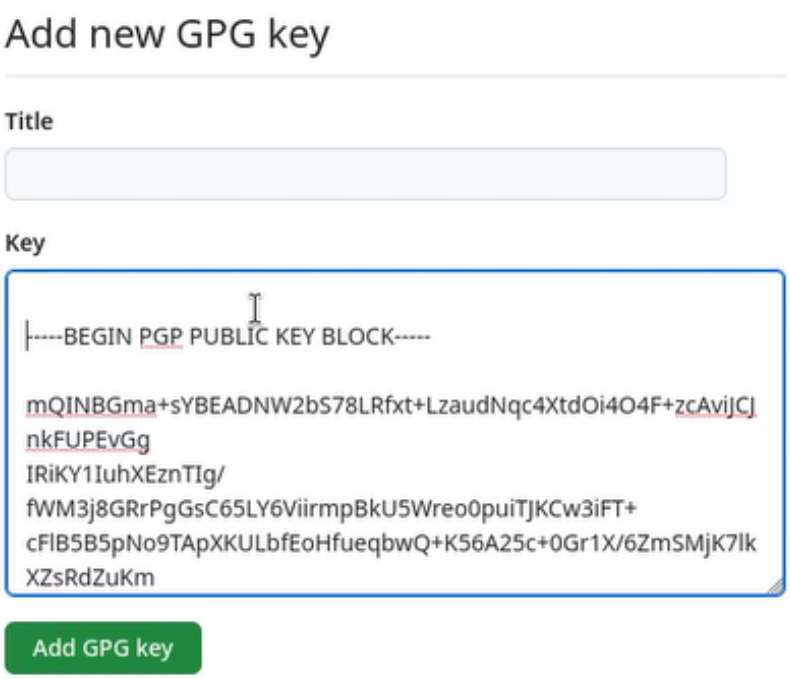

Рисунок 3.8: Ввод ключа

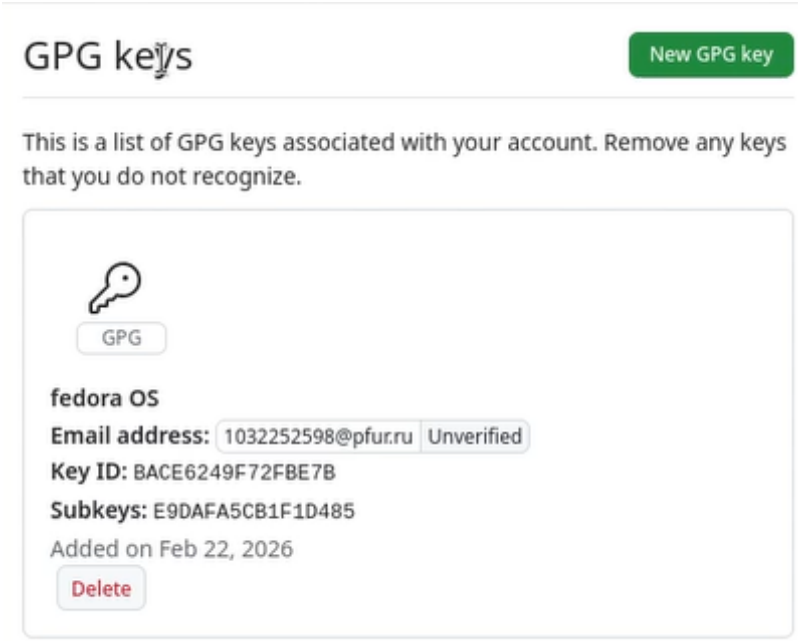

Рисунок 3.9: Готовый ключ

Возвращаемся к базовой настройке git. Задали имя начальной ветки (будем

называть её master) и параметры autocrlf и safecrlf(рис. 3.10).

```
root@aoivanova:~# git config --global init.defaultBranch master
root@aoivanova:~# git config --global core.autocrlf input
root@aoivanova:~# git config --global core.safecrlf warn
```

Рисунок 3.10: Задание названия ветки и параметры autocrlf и safecrlf

3.3 Создание ключей

По алгоритму rsa с ключём размером 4096 бит создали ключ SSH (рис. 3.11).

```
root@aoivanova:~# ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase for "/root/.ssh/id_rsa" (empty for no passphra
se):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa
Your public key has been saved in /root/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:dMlofolLnCb7lNUwA7hpp0O2yzB0dGVbHCgFcTOgWr4 root@aoivano
va
The key's randomart image is:
+---[RSA 4096]----+
|        .=*Boo.  |
|       o.o*.*.   |
|      .o++.O     |
|     .+B=.+ *    |
|    ..=o+S + .   |
|     o +* =      |
|      +Eo+       |
|       oo        |
|         .       |
+----[SHA256]-----+
```

Рисунок 3.11: Создание по алгоритму rsa с ключём размером 4096 бит

По алгоритму ed25519 создали ключ SSH (рис. 3.12).

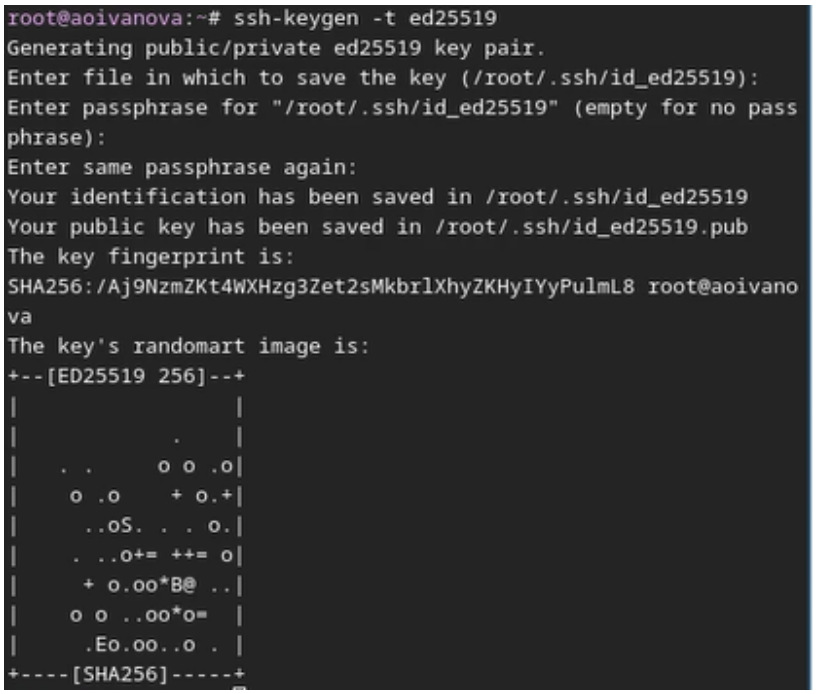

Рисунок 3.12: Создание по алгоритму ed25519

Ключ PGP уже был сознан и добавлен в GitHub во время настройки верификацию и подписание коммитов git.

3.4 Настройка автоматических подписей коммитов gi

Используя введёный email, указали Git применять его при подписи коммитов (рис. 3.13).

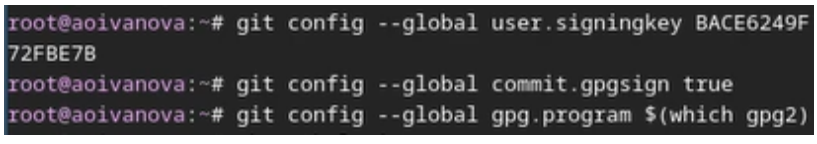

Рисунок 3.13: Создание по алгоритму ed25519

3.5 Настройка gh

Для начала авторизовались с помощью команды gh auth login (рис. 3.14).

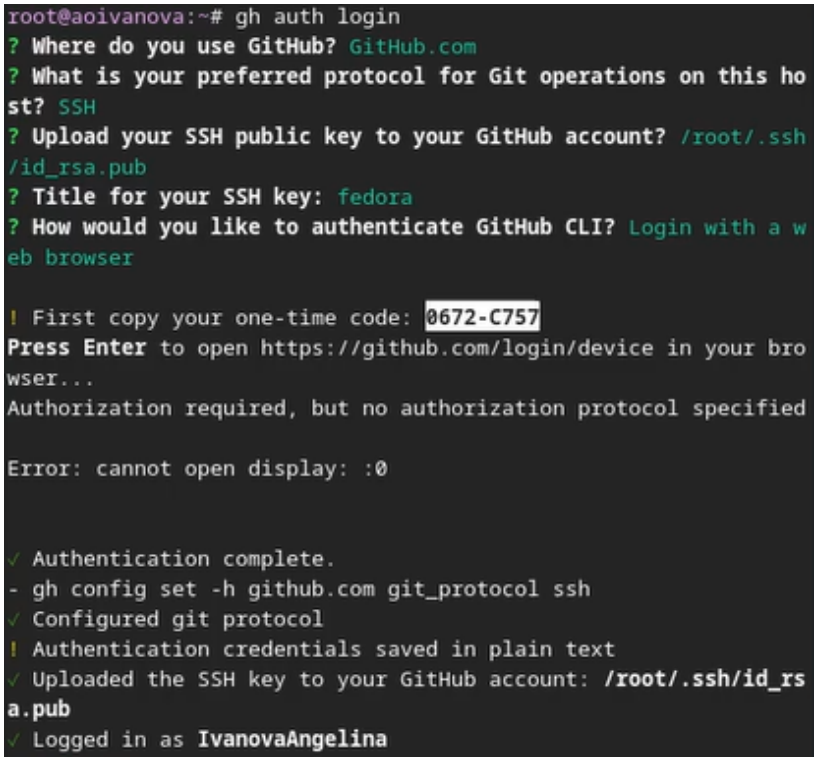

Рисунок 3.14: Авторизация

3.6 Сознание репозитория курса на основе шаблона

Создали по шаблону рабочее пространство. На GitHub я уже создала необходимый репзиторий, поэтому необходимо только клонировать его на свою виртуальную машину я (рис. 3.15).

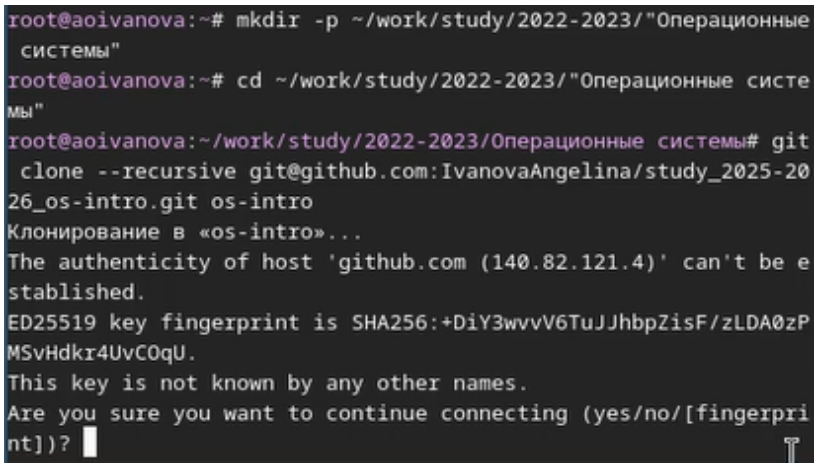

Рисунок 3.15: Создание рабочего пространства

3.7 Настройка каталога курса

Перешли в каталог курса, удалили лишние файлы и создали необходимые каталоги (рис. 3.16).

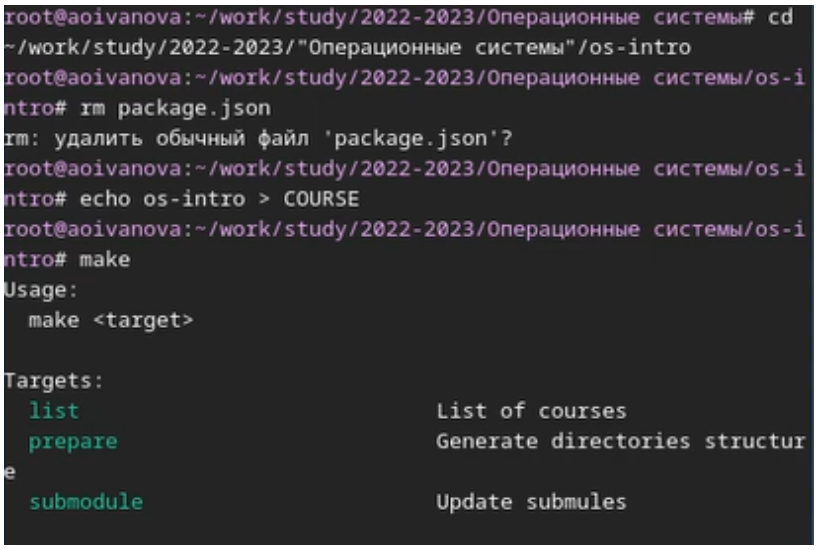

Рисунок 3.16: Создание рабочего пространства

Отправили файлы на сервер (рис. 3.17).

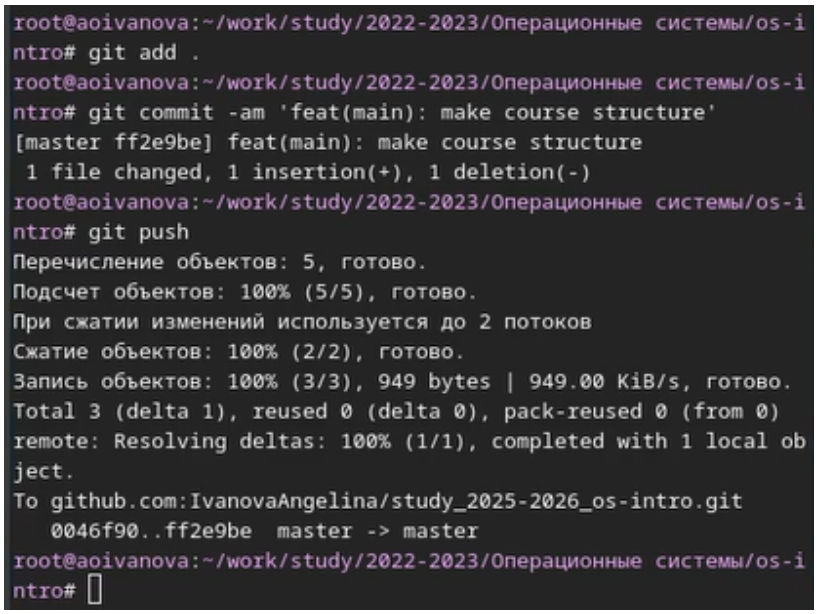

Рисунок 3.17: Создание рабочего пространства

4 Контрольные вопросы

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

Система контроля версий (Version Control System, VCS) представляет собой программное обеспечение, которое позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п.

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

хранилище - это репозиторий, в котором хранятся все файлы и документы, включая историю изменений. commit - отслеживание и сохранение изменений. история - созраняет в себе изменения проекта на всех этапах. рабочая копия - - копия проекта, связанная с репозиторием.

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

Централизованные VCS: В централизованных VCS весь код и его история хранятся в одном центральном репозитории. Разработчики работают с копией основного репозитория на своих локальных машинах, откуда отправляют изменения в центральное хранилище.

Примеры: - Subversion (SVN): Один из популярных централизованных VCS. Разработчики могут коммитить изменения в центральный репозиторий и обновлять свои локальные копии. - Perforce: Еще один пример системы контроля версий с централизованным подходом, который широко применяется в больших коммерческих проектах.

Децентрализованные VCS: Децентрализованные VCS позволяют каждому участнику проекта иметь полноценную копию всего репозитория. Это означает, что разработчики имеют доступ ко всей истории проекта локально и могут работать независимо от подключения к сети.

Примеры: - Git: Самая популярная децентрализованная система контроля версий. Разработчики могут коммитить, откатывать изменения и создавать ветки без необходимости доступа к центральному серверу. - Mercurial: Еще один пример децентрализованной VCS, обеспечивающий высокую скорость работы и гибкость в управлении кодом.

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

При единоличной работе с хранилищем (репозиторием) в системе контроля версий (VCS), разработчик ведет работу над кодом самостоятельно без коллективного взаимодействия. В такой ситуации основной упор делается на сохранение версий кода и отслеживание изменений для личного удобства и безопасности. Вот основные действия, которые могут выполняться при единоличной работе с хранилищем:

- Инициализация репозитория
- Клонирование репозитория
- Коммит изменений
- Просмотр истории изменений
- Создание веток
- Обновление репозитория

- Удаление, перемещение файлов
- Откат изменений
- Игнорирование файлов
- Резервное копировани

5. Опишите порядок работы с общим хранилищем VCS.

Работа с общим хранилищем (репозиторием) в системе контроля версий (VCS) включает в себя совместную работу нескольких разработчиков над одним проектом. Вот порядок действий при работе с общим хранилищем VCS:

- Создание или клонирование репозитория
- Получение последних изменений
- Внесение изменений
- Коммит
- Отправка изменений (push)
- Работа с веткам и т.д.

6. Каковы основные задачи, решаемые инструментальным средством git?

Основные задачи, решаемые инструментальным средством Git, включают в себя управление версиями кода, обеспечение совместной работы над проектами, отслеживание изменений, создание и объединение ветвей разработки, а также возможность отката к предыдущим версиям кода. Git также предоставляет возможность создания резервных копий (backup) и управление изменениями в коде, что делает его ключевым инструментом для разработчиков программного обеспечения.

7. 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
- git diff - просмотр текущих изменений
- 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 доба- вленные 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

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

Git pull: Команда git pull используется для извлечения и загрузки содержимого из удаленного репозитория и немедленного обновления локального репозитория

этим содержимым. Слияние удаленных вышестоящих изменений в локальный репозиторий — это обычное дело в процессе совместной работы на основе Git.

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

Ветви (branches) в контексте систем контроля версий, таких как Git, представляют собой параллельные линии разработки, которые позволяют команде разработчиков работать над отдельными фрагментами кода независимо друг от друга. Ветви могут быть полезны для разработки новых функций, исправления ошибок, экспериментов с кодом и поддержания стабильной основной версии программного обеспечения. Создание и использование ветвей помогает упростить процесс разработки, избегая конфликтов и обеспечивая возможность параллельной работы над различными задачами.

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

Игнорируемый файл — файл, явным образом помеченный для Git как файл, который необходимо игнорировать.

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. Вот некоторые распространенные примеры таких файлов (например: /bin, .lock, .tmp, /packages)

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 Выводы

В ходе выполнения лабораторной работы мы изучили идеологию и примененинние средств контроля версий, а также освоили умения по работе с git.

6 Список литературы

1. Лабораторая работа №2 [Электронный ресурс] URL: https://esystem.rudn.ru/mod/page/view.php
2. GitHub [Электронный ресурс] URL: https://github.com/