**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

**ОТЧЕТ  
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

*дисциплина: Операционные системы*

**Студент:** Кашкин Иван

**Группа:** НБИбд-01-21

**Ст. билет №:** 1032212958

Москва

2022 г.

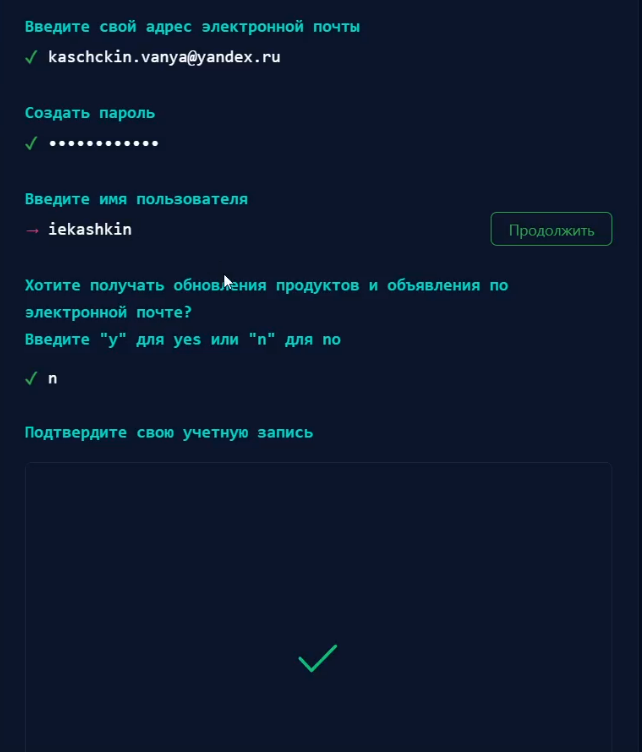
**Цель:**

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

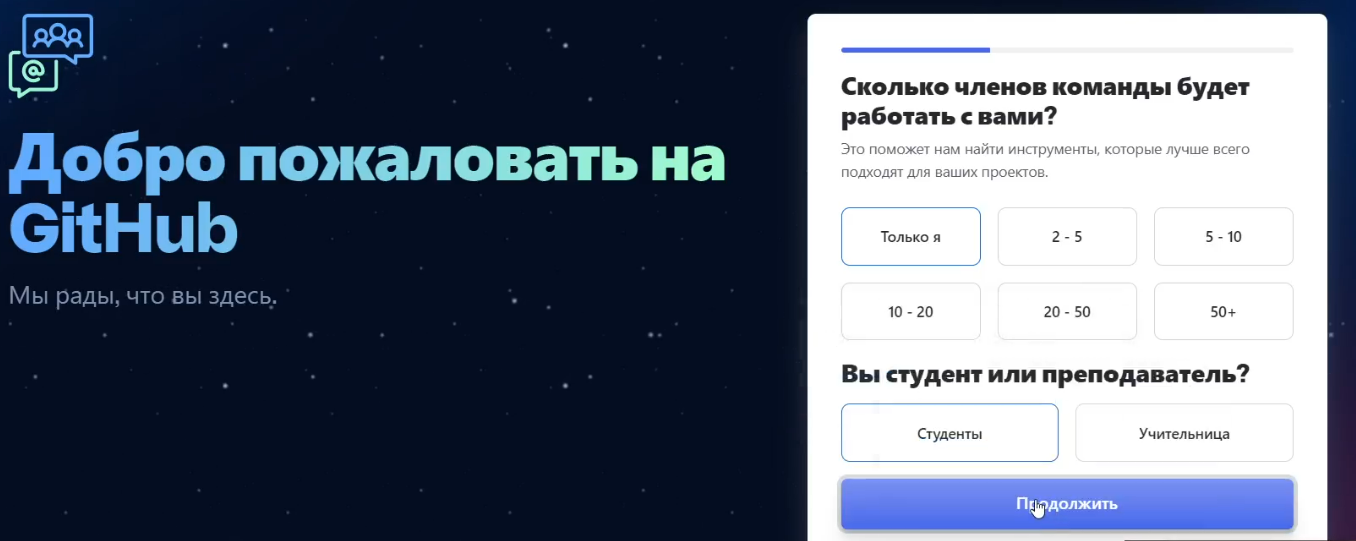
**Ход работы:**

1. Настройка GitHub:

* Создайте учётную запись на https://github.com.
* Заполните основные данные на <https://github.com> (рис. 1.1; рис 1.2)



***(Рис.1.1)***

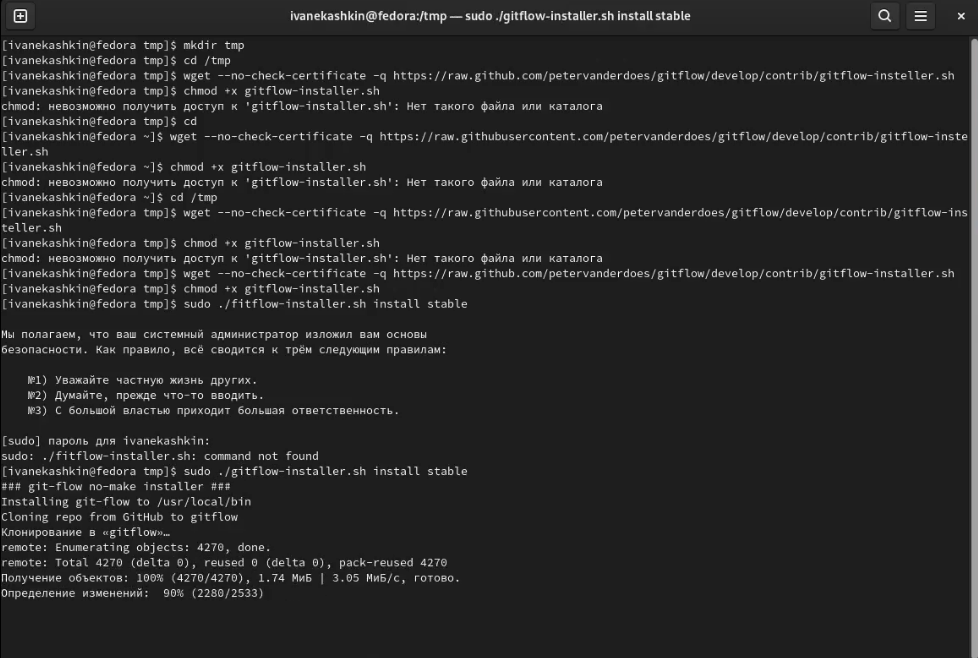


***(Рис. 1.2)***

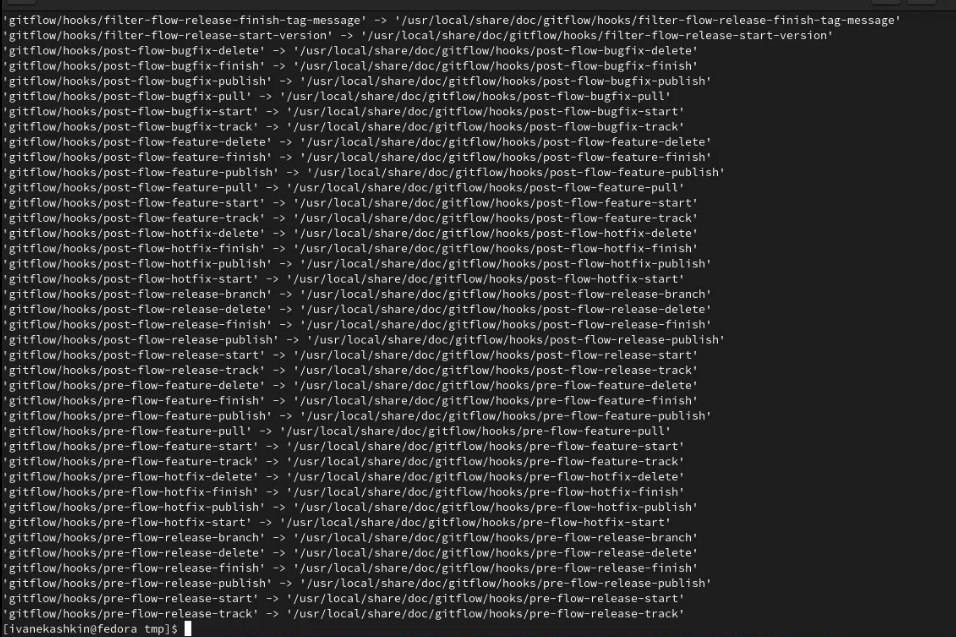
1. Установка программного обеспечения:

Мы начали выполнять этот пункт с установки git-flow н нашу виртуальную машину. Это программное обеспечение удалено из репозитория. Необходимо устанавливать его вручную: (рис. 2.1; рис. 2.2)

1. «cd /tmp»
2. «wget --no-check-certificate -q <https://raw.github.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh>»
3. «chmod +x gitflow-installer.sh»
4. «sudo ./gitflow-installer.sh install stable»



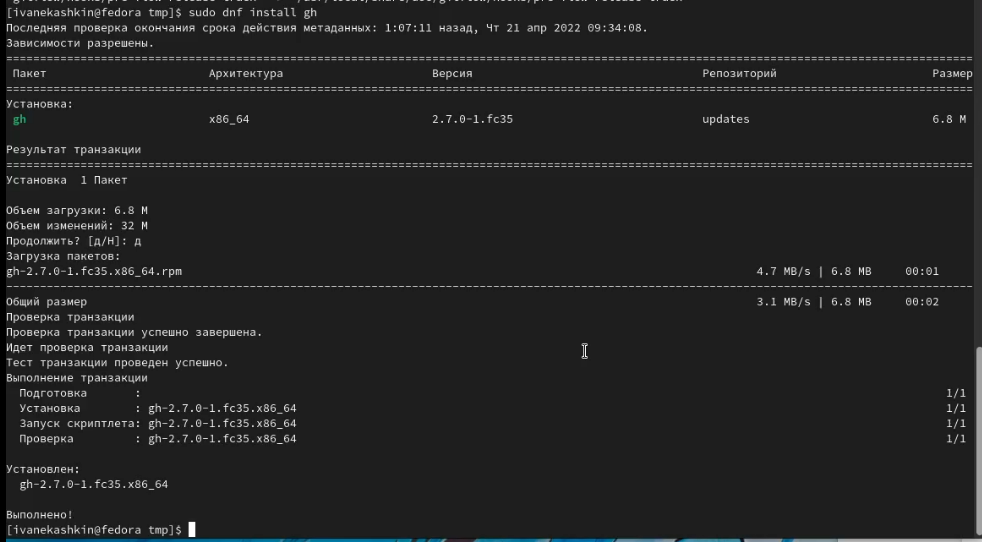
***(Рис.2.1)***



***(Рис.2.2)***

После этого нам нужно установить gh в Fedora Linux с помощью команды:

«sudo dnf install gh»(рис. 3)



***(Рис. 3)***

Далее по лабораторной работе мы устанавливали базовые настройки git с помощью программ: (рис. 4)

1. «git config --global user.name "Name Surname"»
2. «git сonfig --global user.email "work@mail"»

Эти две команды выше задают имя и email владельца репозитория.

Настроим utf-8 в выводе сообщений git с помощью: (рис.4)

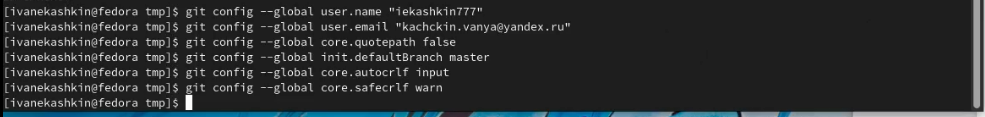
1. «git config --global core.quotepath false»

Настроим верификацию и подписание коммитов git. Зададим имя начальной ветки master: (рис. 4)

1. «git сonfig --global init.defaultBranch master»

Настраиваем параметры autocrlf и safecrlf: (рис. 4)

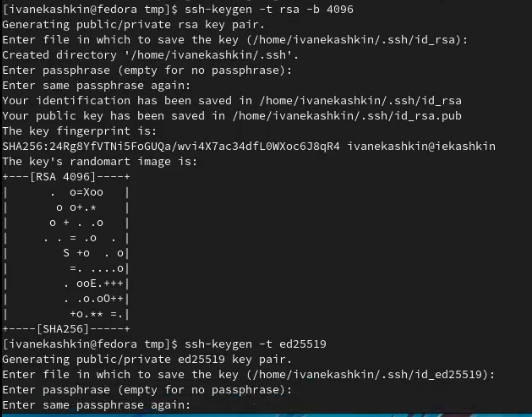
1. «git config --global core.autocrlf input»
2. «git config --global core.safecrlf warn»



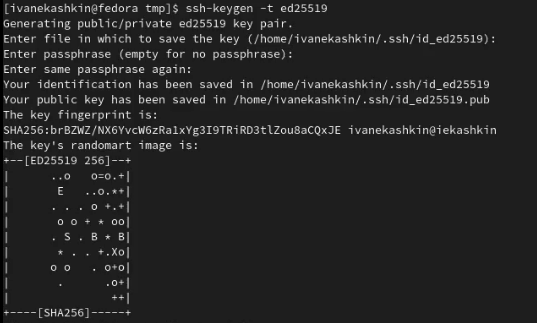
***(Рис. 4)***

От этих действий мы перешли к созданию ключа ssh. По алгоритму rsa с ключём размером 4096 бит, а после по алгоритму ed25519, пишем комнды: (рис.5.1; рис.5.2)

1. «ssh-keygen -t rsa -b 4096»
2. «ssh-keygen -t ed25519»



***(Рис.5.1)***



***(Рис.5.2)***

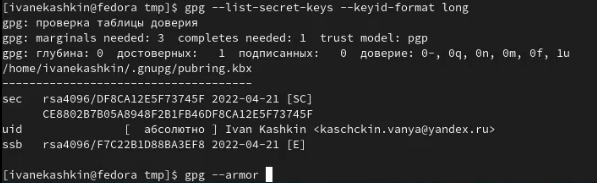
После создаем ключ gpg: (рис.6)

1. «gpg --full-generate-key»

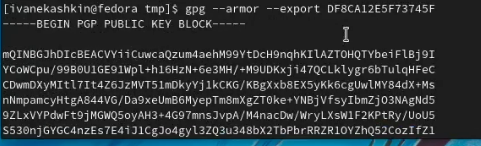


***(Рис.6)***

Добавим этот ключ в GitHub (рис.7.1 рис.7.2 рис.7.3 рис.7.4) (<PGP Fingerprint> | xclip -sel clip – Эта часть команды у меня не работал и я писал свой «Отпечаток\_ключа»)



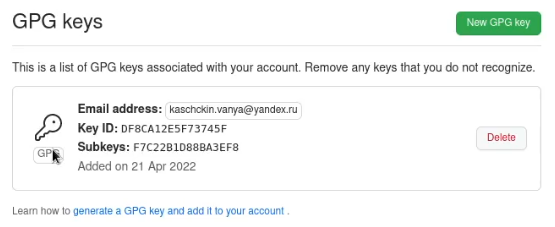
***(Рис.7.1)***



***(Рис.7.2)***



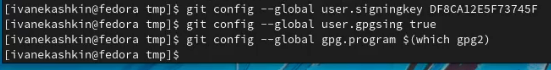
***(Рис.7.3)***



***(Рис.7.4)***

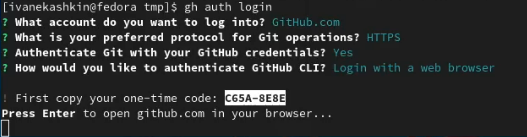
Далее идет настройка настройка автоматических подписей коммитов git. Мы используем введёный email, укажем Git применять его при подписи коммитов: (рис.8)

1. «git config --global user.signingkey «Отпечаток\_ключа»»
2. «git config --global commit.gpgsign true»
3. «git config --global gpg.program $(which gpg2)»

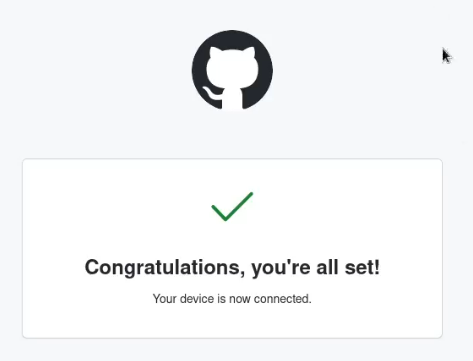


***(Рис.8)***

Настройка gh и авторизация с помошью консоли (рис.9.1 рис9.2)



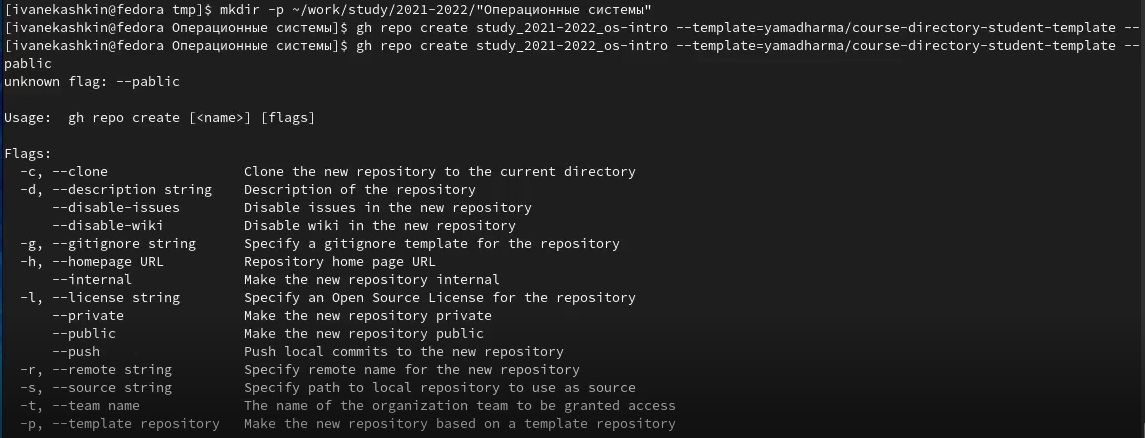
***(Рис.9.1)***



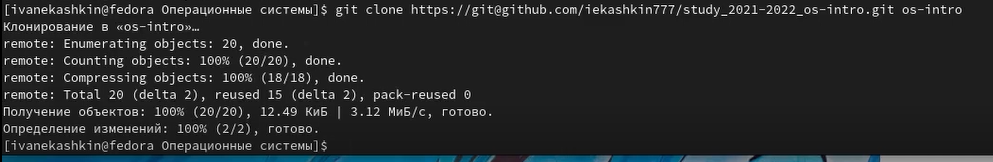
***(Рис.9.2)***

Мы создаем репозиторий курса на основе шаблона (рис.10.1 рис.10.2)

1. mkdir -p ~/work/study/2021-2022/"Операционные системы"
2. cd ~/work/study/2021-2022/"Операционные системы"
3. gh repo create study\_2021-2022\_os-intro --template=yamadharma/course-directory-student-template --public
4. git clone --recursive git@github.com:<iekashkin777>/study\_2021-2022\_os-intro.git os-intro



***(Рис.10.1)***



***(Рис.10.2)***

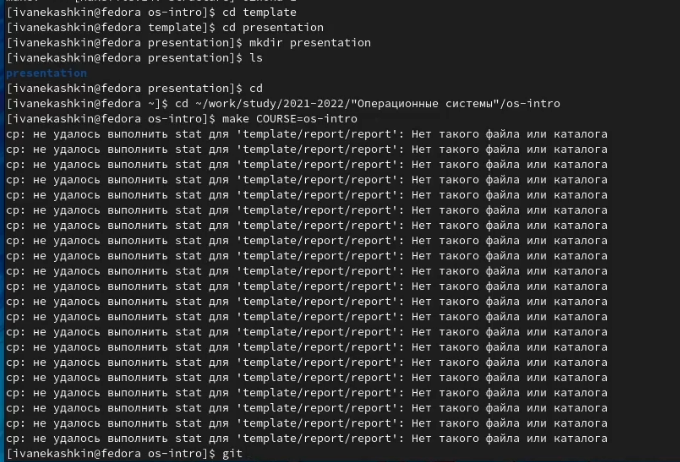
И в конце идет настройка каталога курса (рис11.1 рис 11.2 рис 11.3 рис 11.4)

1. cd ~/work/study/2021-2022/"Операционные системы"/os-intro
2. rm package.json
3. make COURSE=os-intro

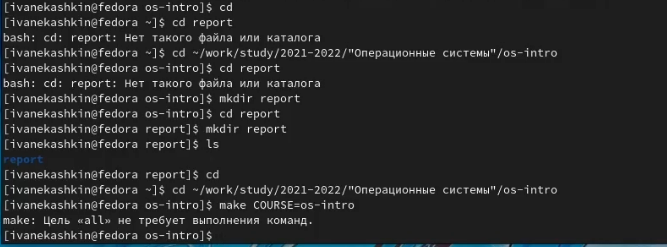
Поеснение для этих команд: первая, мы заходим в каталог курса, вторая, удаляем не нужные файлы, третья, создаем необходимые каталоги

******

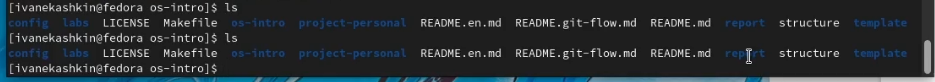
***(Рис.11.1)***

******

***(Рис.11.2)***

******

***(Рис.11.3)***

******

***(Рис.11.4)***

**Контрольные вопросы:**

1) Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. Системы контроля версий (Version Control System, VCS)применяются при работе нескольких человек над одним проектом.

2) 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 неполную версию изменённых файлов, а производить так называемую дельта-компрессию—сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных.

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.

3) Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. Пример - Wikipedia.

В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. Пример — Bitcoin.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

4) Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

git config --global user.name"Имя Фамилия"

git config --global user.email"work@mail"

и настроив utf-8 в выводе сообщений git:

git config --global quotepath false

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

cd

mkdir tutorial

cd tutorial

git init

5) Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C"Имя Фамилия <work@mail>"

Ключи сохраняться в каталоге~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

cat ~/.ssh/id\_rsa.pub | xclip -sel clip вставляем ключ в появившееся на сайте поле.

6) У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

7) Основные команды git:

Наиболее часто используемые команды git: – создание основного дерева репозитория :git init–получение обновлений (изменений) текущего дерева из центрального репозитория: git pull–отправка всех произведённых изменений локального дерева в центральный репозиторий:git push–просмотр списка изменённых файлов в текущей директории: git status–просмотр текущих изменения: git diff–сохранение текущих изменений:–добавить все изменённые и/или созданные файлы и/или каталоги: git add .–добавить конкретные изменённые и/или созданные файлы и/или каталоги: git add имена\_файлов – удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): git rm имена\_файлов – сохранение добавленных изменений: – сохранить все добавленные изменения и все изменённые файлы: git commit -am 'Описание коммита'–сохранить добавленные изменения с внесением комментария через встроенный редактор: git commit–создание новой ветки, базирующейся на текущей: git checkout -b имя\_ветки–переключение на некоторую ветку: git checkout имя\_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: git push origin имя\_ветки–слияние ветки стекущим деревом:git merge --no-ff имя\_ветки–удаление ветки: – удаление локальной уже слитой с основным деревом ветки:git branch -d имя\_ветки–принудительное удаление локальной ветки: git branch -D имя\_ветки–удаление ветки с центрального репозитория: git push origin :имя\_ветки

8) Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

git add hello.txt

git commit -am 'Новый файл’

9) Проблемы, которые решают ветки git:

• нужно постоянно создавать архивы с рабочим кодом

• сложно "переключаться" между архивами

• сложно перетаскивать изменения между архивами

• легко что-то напутать или потерять

10) Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл.gitignore с помощью сервисов. Для этого сначала нужно получить списоки меняющихся шаблонов: curl -L -s https://www.gitignore.io/api/list

Затем скачать шаблон, например, для C и C++

curl -L -s https://www.gitignore.io/api/c >> .gitignore

curl -L -s https://www.gitignore.io/api/c++ >> .gitignore

**Вывод:**

Я изучил изучить идеологию и применение средств контроля версий. Освоил умения по работе с git.