ЛАБОРАТОРНАЯ РАБОТА № 2
НАСТРОЙКА СИСТЕМЫ УПРАВЛЕНИЯ ВЕРСИЯМИ GIT

Время выполнения – 2 часа.

Цель работы: приобретение навыков использования систем контроля версий Git.

Задачи работы

1. Ознакомиться этапами установки и настройки git;
2. Научиться:
- устанавливать git;
- выполнять инициализацию каталога для работы в PyCharm.

Перечень обеспечивающих средств

Для выполнения работы необходимо иметь компьютер с установленной операционной системой семейства Windows, установленным python и IDE PyCharm Professional.

Общие теоретические сведения

Система управления версиями (Version Control System, VCS) – программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Однако они могут с успехом применяться и в других областях, в которых ведётся работа с большим количеством непрерывно изменяющихся электронных документов. В частности, системы управления версиями применяются в САПР, обычно в составе систем управления данными об изделии. Управление версиями используется в инструментах конфигурационного управления (Software Configuration Management Tools).

Виды VCS:

– локальные системы контроля версий;

– централизованные системы контроля версий;

– децентрализованные системы контроля версий.

Локальные системы контроля версий.

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий.

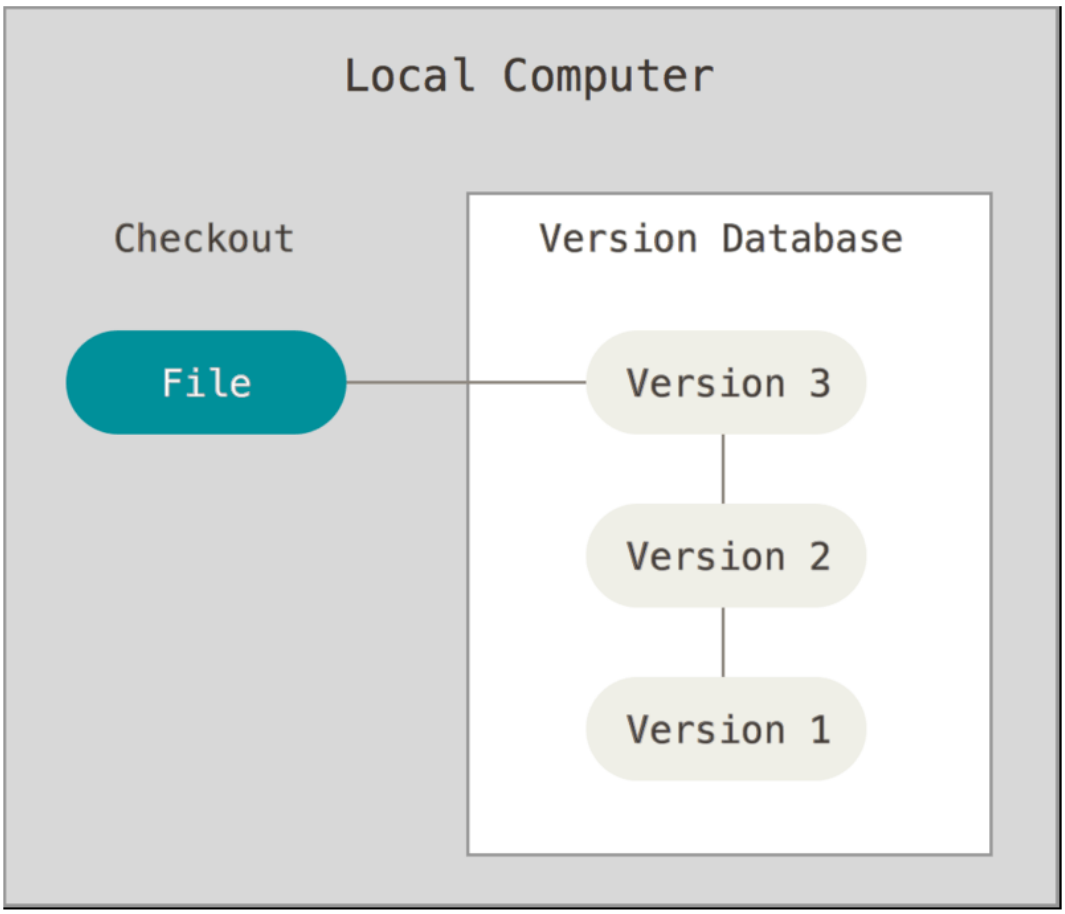

Рисунок 2.1 – Локальный контроль версий

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. Даже популярная операционная система Mac OS X предоставляет команду rcs, после установки Developer Tools. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

Централизованные системы контроля версий.

Следующая серьёзная проблема, с которой сталкиваются люди – это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как: CVS, Subversion и Perforce, имеют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного

хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет.

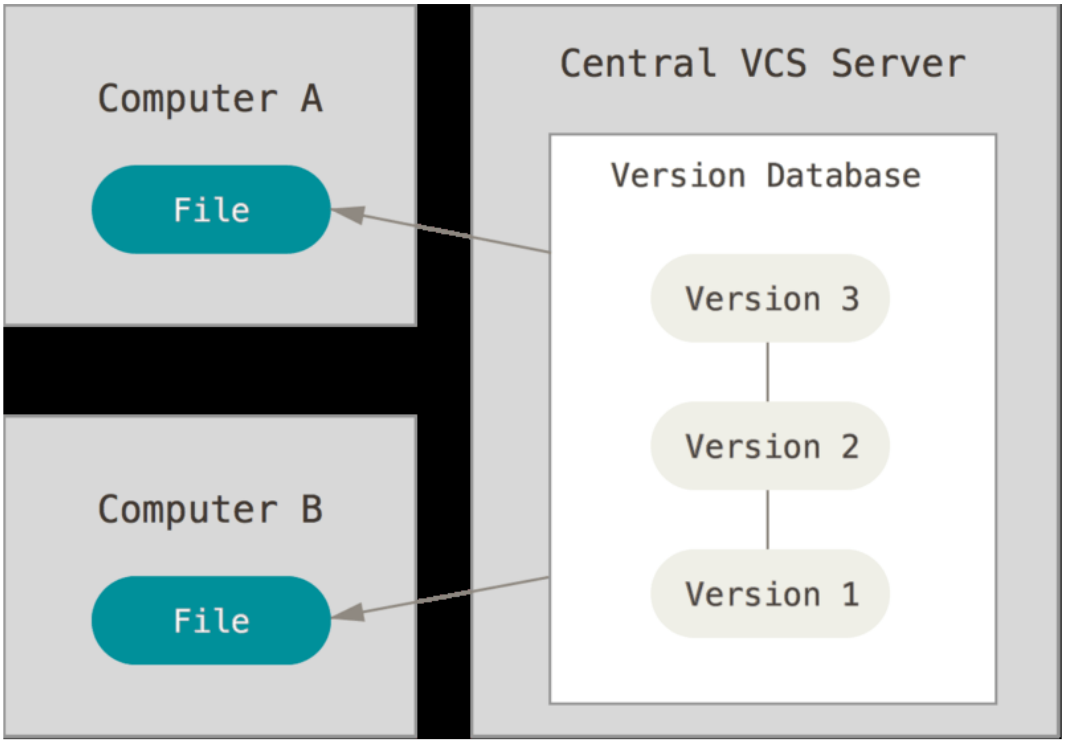

Рисунок 2.2 – Централизованный контроль версий

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус – это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми он работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё – всю историю проекта, не считая

единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы – когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

Децентрализованные системы контроля версий

Здесь в игру вступают децентрализованные системы контроля версий (ДСКВ). В ДСКВ (таких как Git, Mercurial, Bazaar или Darcs), клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени): они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных.

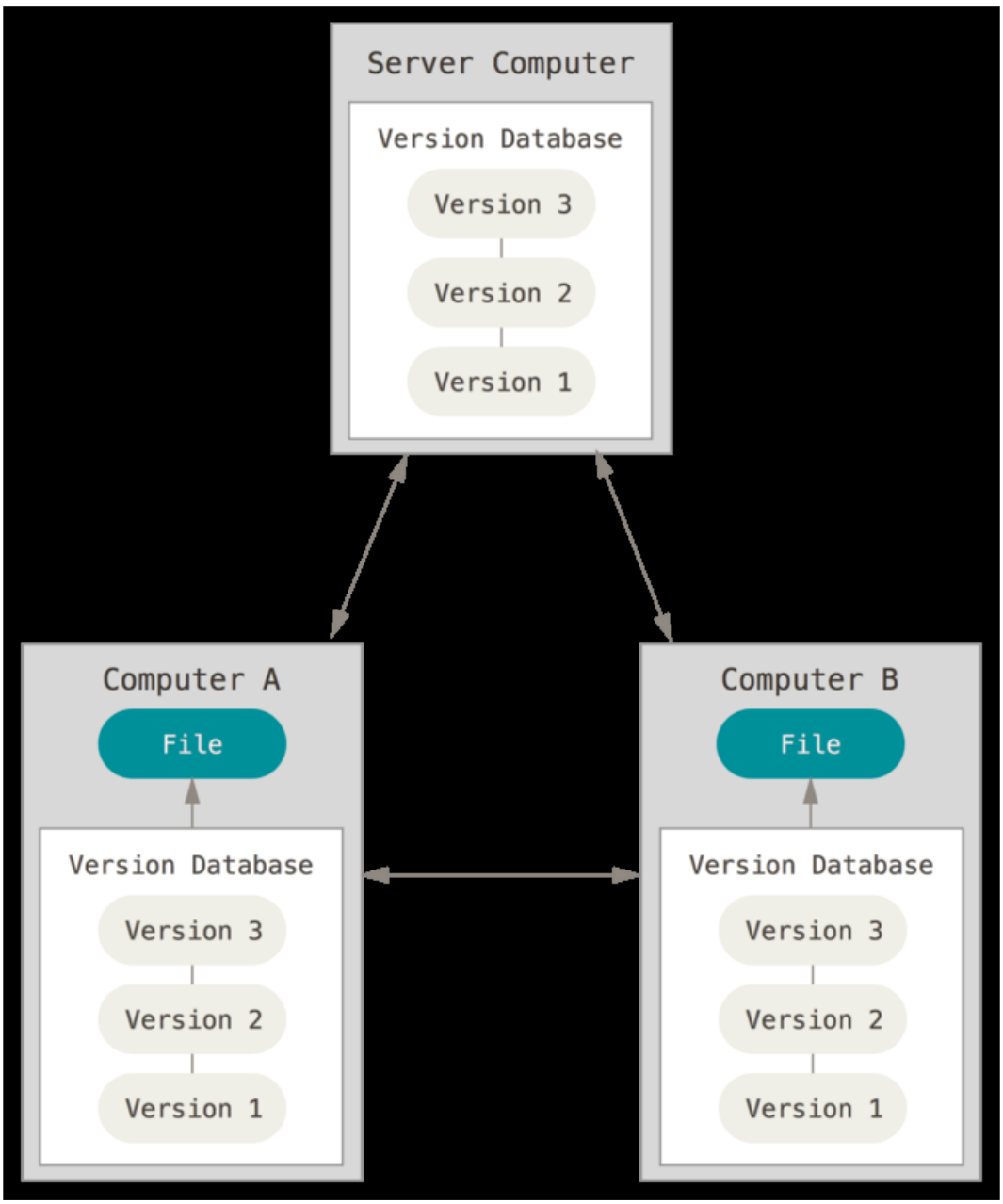

Рисунок 2.3 – Децентрализованный контроль версий

Более того, многие ДСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно, в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

Задание

1. Зарегистрироваться на сервисе GitHub.
2. Используя средства для работы с Git встроенные в PyCharm:

– Создать локальный репозиторий.

– Создать в локальном репозитории приложение, которое выводит в консоль ФИО студента, факультет, курс, группу и email.

– Синхронизировать локальный репозиторий, с созданным на GitHub.

Технология выполнения работы

PyCharm позволяет управлять проектами Git, размещенными на GitHub, непосредственно из IDE: клонировать репозитории, делиться проектами, создавать ветки, делиться кодом через сущности, создавать запросы на вытягивание и просматривать входящие запросы на вытягивание.

Чтобы иметь возможность получать данные из репозитория, размещенного на GitHub, или делиться своими проектами, вам необходимо зарегистрировать свою учетную запись GitHub в PyCharm.

Для создания нового аккаунта нажмите *Ctrl+Alt+S*, чтобы открыть настройки IDE, и выберите *Version Control | GitHub* (Рисунок 2.1).

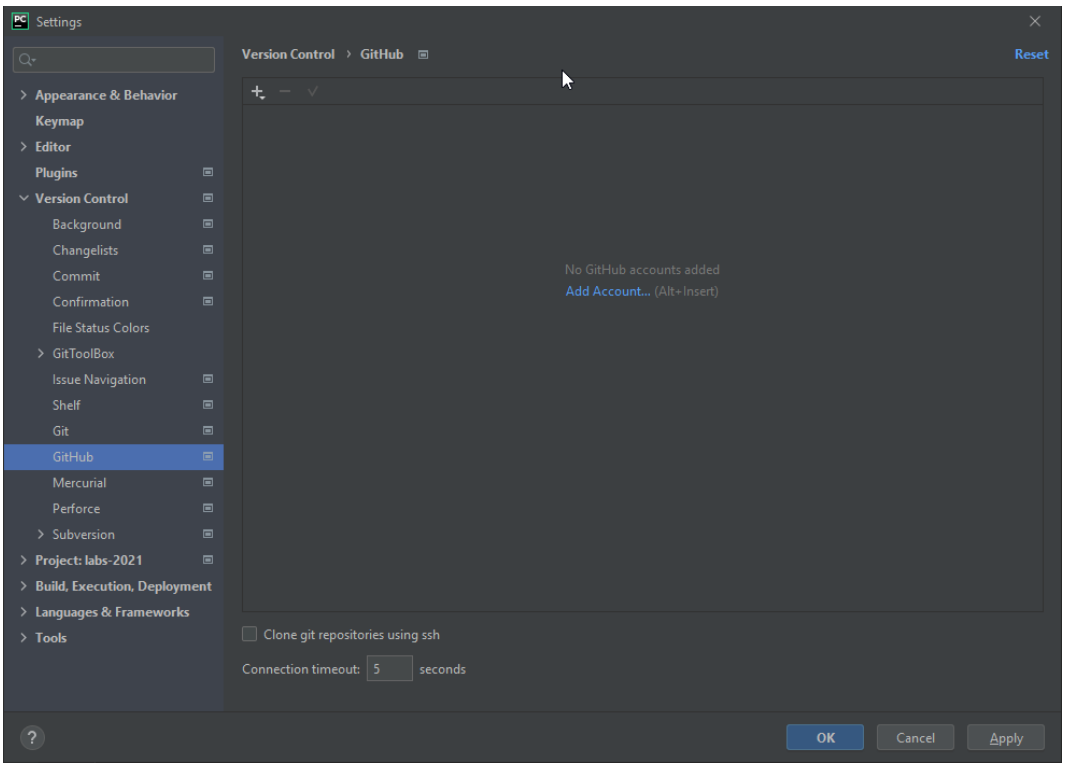

Рисунок 2.1 – Настройки IDE

Далее необходимо нажать *Add account* и в открывшемся диалоговом окне выбрать *Sign up for Github*. Зарегистрируйте свою учетную запись на открывшейся странице регистрации GitHub. Далее вернитесь к настройкам PyCharm, нажмите кнопку + и выберите пункт Log in with Token. Появится окно, представленное на рисунке 2.2.

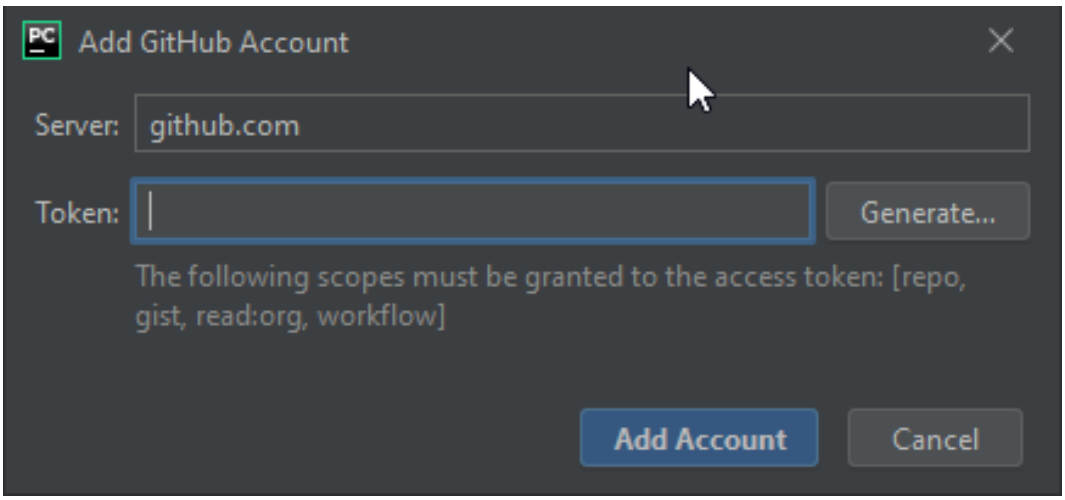

Рисунок 2.2. – Добавление аккаунта GitHub

Необходимо создать личный токен доступа, который будет использоваться вместо пароля в командной строке или API. Токены личного доступа (PAT) являются альтернативой использованию паролей для аутентификации в GitHub при использовании GitHub API или командной строки.

Для создания токена нужно выполнить следующие шаги:

3. Подтвердить свой адрес электронной почты, если он еще не подтвержден.

4. В правом верхнем углу любой страницы нажмите фото своего профиля, затем нажмите *Settings* (Рисунок 2.3)

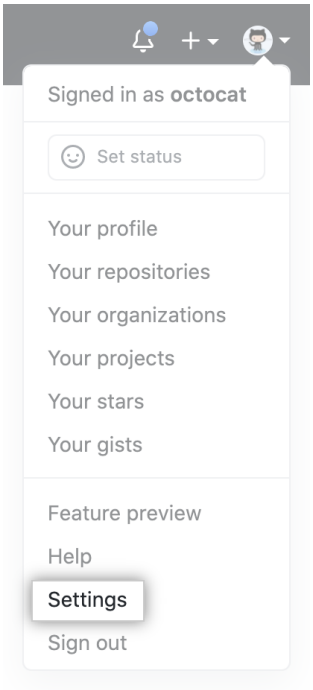

Рисунок 2.3 – Настройки GitHub

5. На левой боковой панели нажать *Developer settings* (Рисунок 2.4*)*

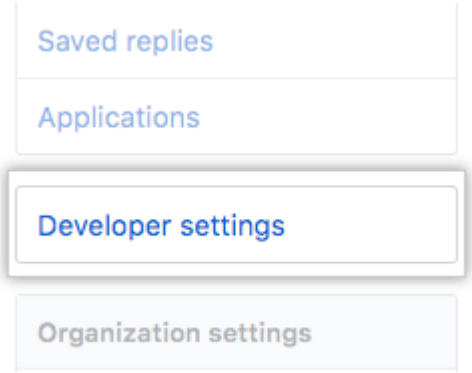

Рисунок 2.4 – Developer settings

6. На левой боковой панели нажать *Personal access tokens* (Рисунок 2.5)

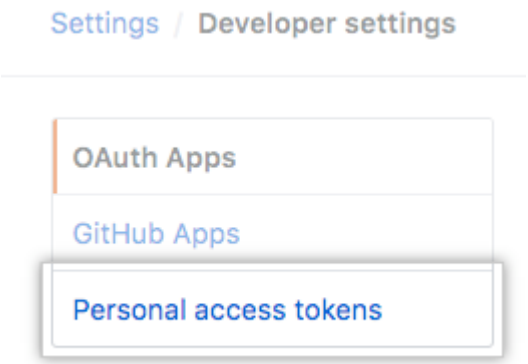

Рисунок 2.5 – Personal access tokens

7. Щелкните *Generate new token* (Рисунок 2.6).

Рисунок 2.6 – Generate new token

8. Дайте вашему токену описательное имя.

9. Чтобы указать срок действия вашего токена, выберите раскрывающееся меню *Expiration*, затем щелкните значение по умолчанию или воспользуйтесь средством выбора календаря.

10. Выберите области или разрешения, которые вы хотите предоставить этому токену. Чтобы использовать свой токен для доступа к репозиториям из командной строки, выберите *repo* (Рисунок 2.7).

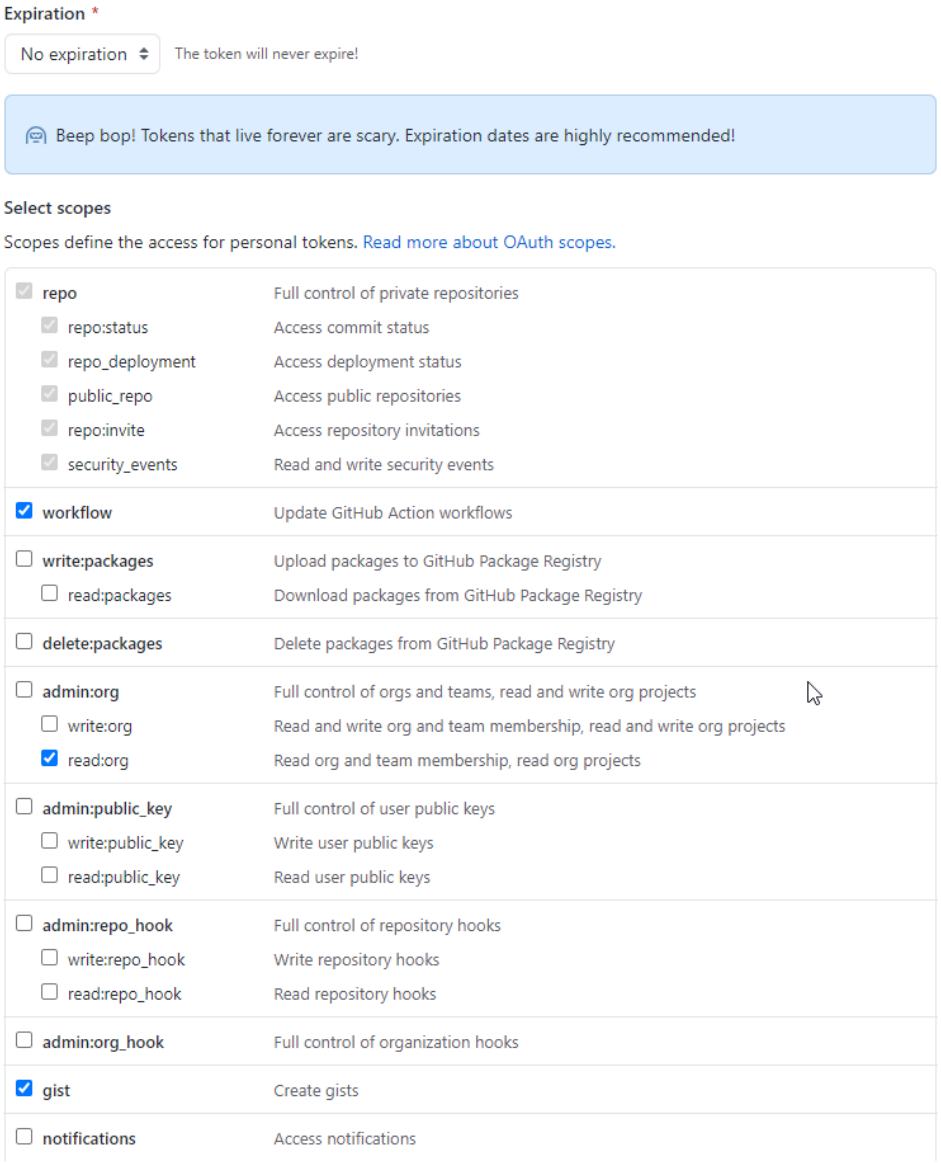

Рисунок 2.7 – Выбор настроек токена

Для токена должны быть включены области *read:org*, *repo*, *gist*, *workflow* в разрешениях вашей учетной записи.

11. Щелкните *Generate token* (Рисунок 2.8)

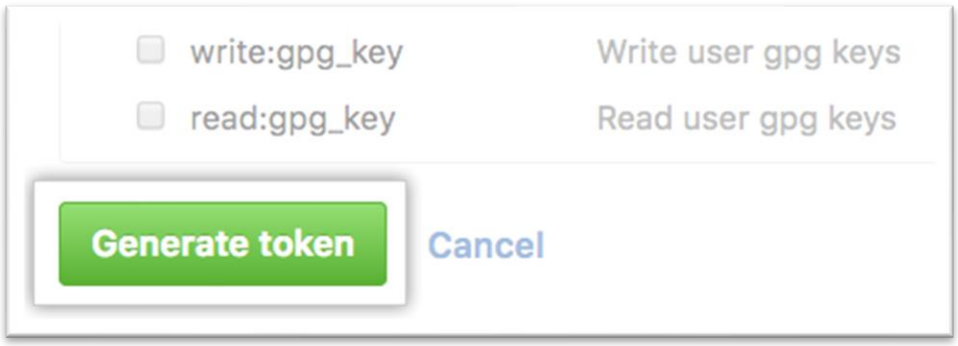

Рисунок 2.8 – Завершение генерации токена

Далее введите сгенерированный токен для активации аккаунта (Рисунок 2.9)

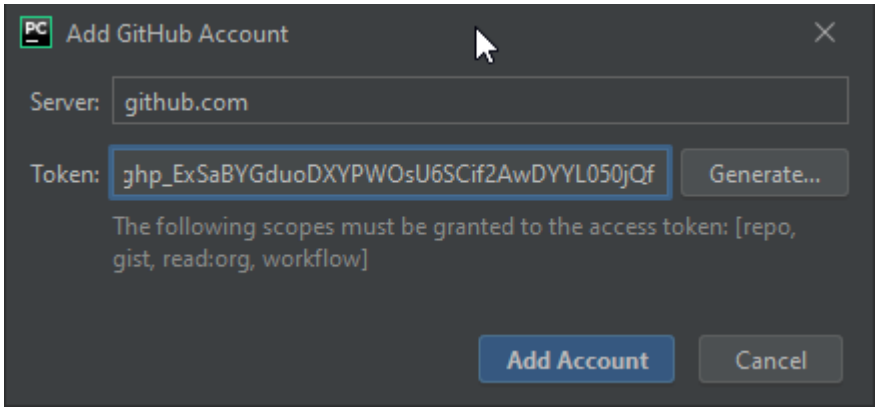

Рисунок 2.9 – Добавление аккаунта с помощью токена

Далее добавить удаленный репозиторий GitHub для проекта, который разрабатывается локально, чтобы другие могли его просматривать или вносить свой вклад. Для этого необходимо:

1. Открыть проект;
2. В главном меню выберите *VCS | Share Project on GitHub*. Если вы уже зарегистрировали свою учетную запись GitHub в PyCharm, соединение будет установлено с использованием этих учетных данных. Если вы не зарегистрировали свою учетную запись в PyCharm, откроется диалоговое окно « Вход в GitHub ». Укажите свой токен доступа или запросите новый с вашим логином и паролем.

3. Когда соединение с GitHub будет установлено, откроется диалоговое окно «Поделиться проектом в GitHub». Укажите новое имя репозитория, имя пульта дистанционного управления и введите описание вашего проекта (Рисунок 2.10).

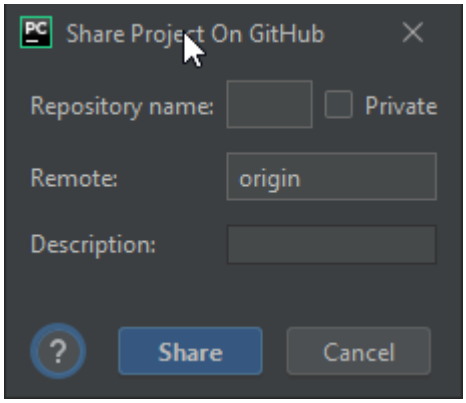

Рисунок 2.10 – Соединение с GitHub

4. Вы можете выбрать опцию *Private*, если не хотите разрешать публичный доступ к вашему репозиторию другим пользователям GitHub.

5. Нажмите «Share», чтобы создать новый репозиторий и загрузить в него исходники проекта (Рисунок 2.11)

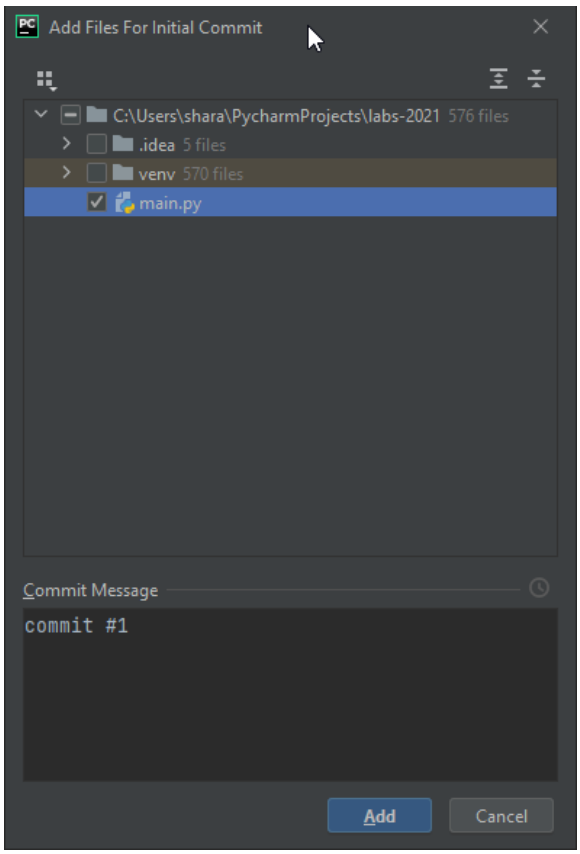

Рисунок 2.11– Выбор файлов для загрузки в репозиторий

Вопросы и задания для защиты работы

1. Что такое git?
2. Дайте понятие репозиторий?
3. Что такое commiting?
4. Что такое branching?
5. Что такое merging?
6. Для чего нужен git?
7. Зачем используют совместную разработку программного обеспечения?
8. В каких состояниях может находиться файл в репозитории?
9. Как происходит изменение состояния файла?
10. Что такое GitHub?