# **Разработка лабораторного практикума**

## **Лабораторная работа №1. Управление проектами и отслеживание ошибок**

### **Основные сведения**

Целью данной лабораторной работы является обучить студента работе с системой управления проектами и системой отслеживания ошибок.

Лабораторная работа №1 будет выполняться с использованием YouTrack.

Студент будет разрабатывать проект согласно индивидуальному заданию.

Студент будет создавать пользователя, создавать и настраивать проект, создавать доску для управления задачами в проекте, создавать задачи и подзадачи, создавать панель мониторинга, создавать отчёты, будет работать со строкой поиска.

**Общие понятия о системе управления проектами и системе отслеживания ошибок**

**Программное обеспечение для управления проектами** – приложение, в котором осуществляются следующие функции: планировка задач, создание расписания, управление бюджетом, совместная работа.

**Система отслеживания ошибок** – программа, целью которой является помочь разработчикам в учете и контроле ошибок, которые были найдены в программах. Также она помогает учитывать пожелания пользователей, следить за процессом устранения ошибок.

**Управление проектом и отслеживание ошибок с помощью YouTrack**

Взаимодействие с YouTrack происходит через браузер. Для того чтобы начать работу с YouTrack необходимо перейти по адресу, настроенному в процессе установки.

Например: 127.0.0.1:81 или localhost:81.

YouTrack обладает дружелюбным пользовательским интерфейсом (Рис. 12). В нем все находится под рукой.

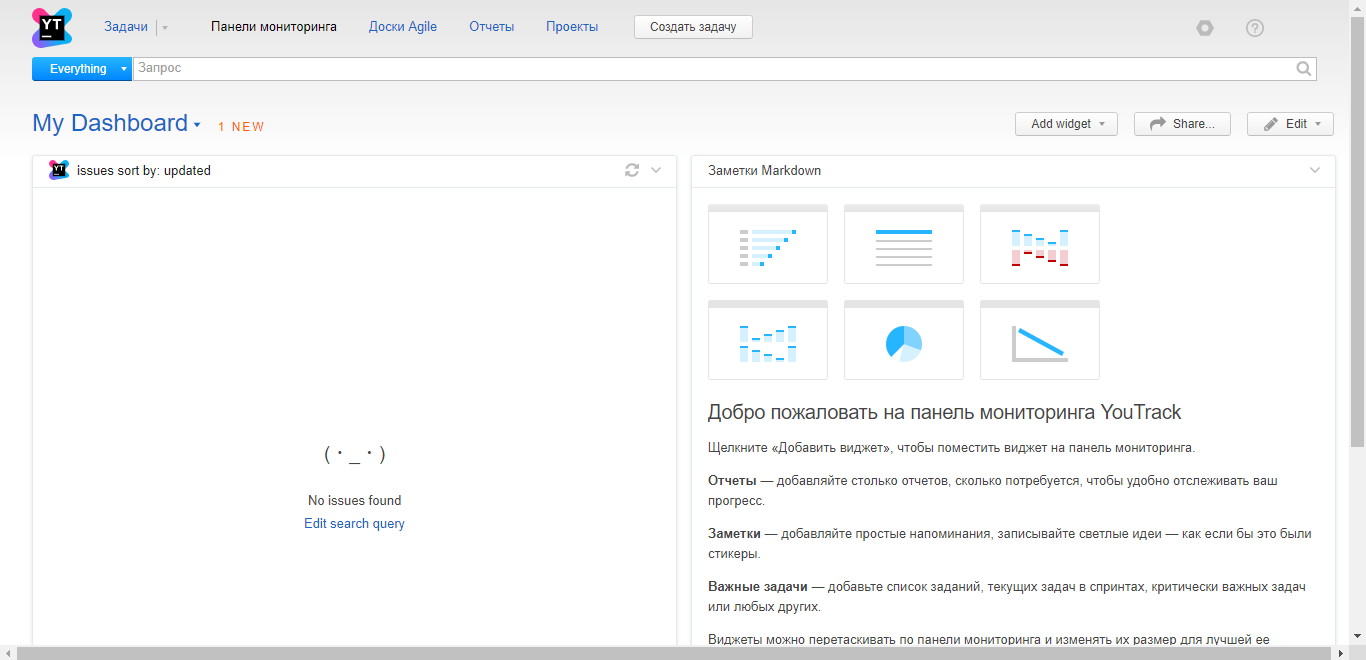


Рисунок 12 – Интерфейс YouTrack

**Создание пользователя в системе YouTrack**

Система управления проектами и система отслеживания ошибок YouTrack является многопользовательской. В ней может работать большое количество пользователей. Они могут обладать различными ролями, правами доступа. Также пользователи могут объединяться в группы. Пользователи могут общаться между собой. Назначать друг друга исполнителями различных задач.

Первый пользователь создаётся при установке системы, также существует пользователь с именем **Гость**. Он подразумевает под собой всех незарегистрированных пользователей, просматривающих проект. Для создания последующих пользователей необходимо сделать следующее:

Необходимо перейти на вкладку **Администрирование** и из выпадающего списка выбрать вкладку **Пользователи** (Рис. 13).

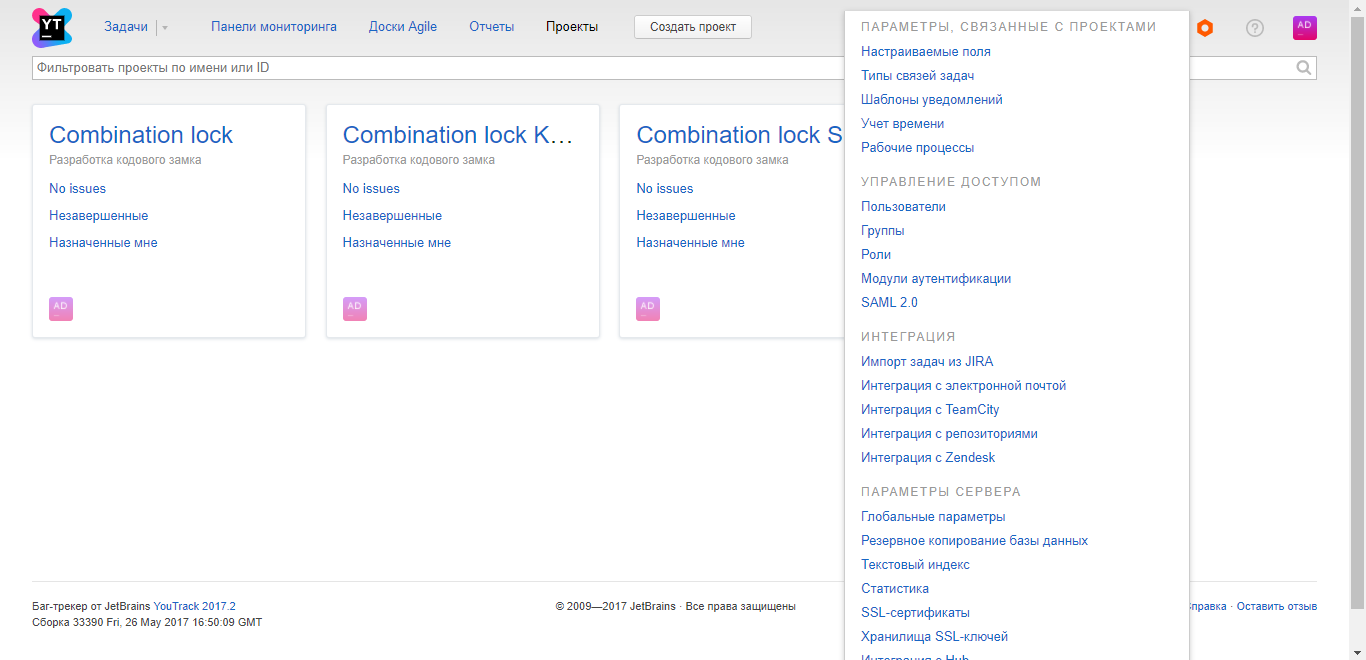


Рисунок 13 – Переход на вкладку пользователи

После этого нужно нажать на кнопку **Новый пользователь** и заполнить форму регистрации, указав имя, e-mail, пароль (Рис. 14).

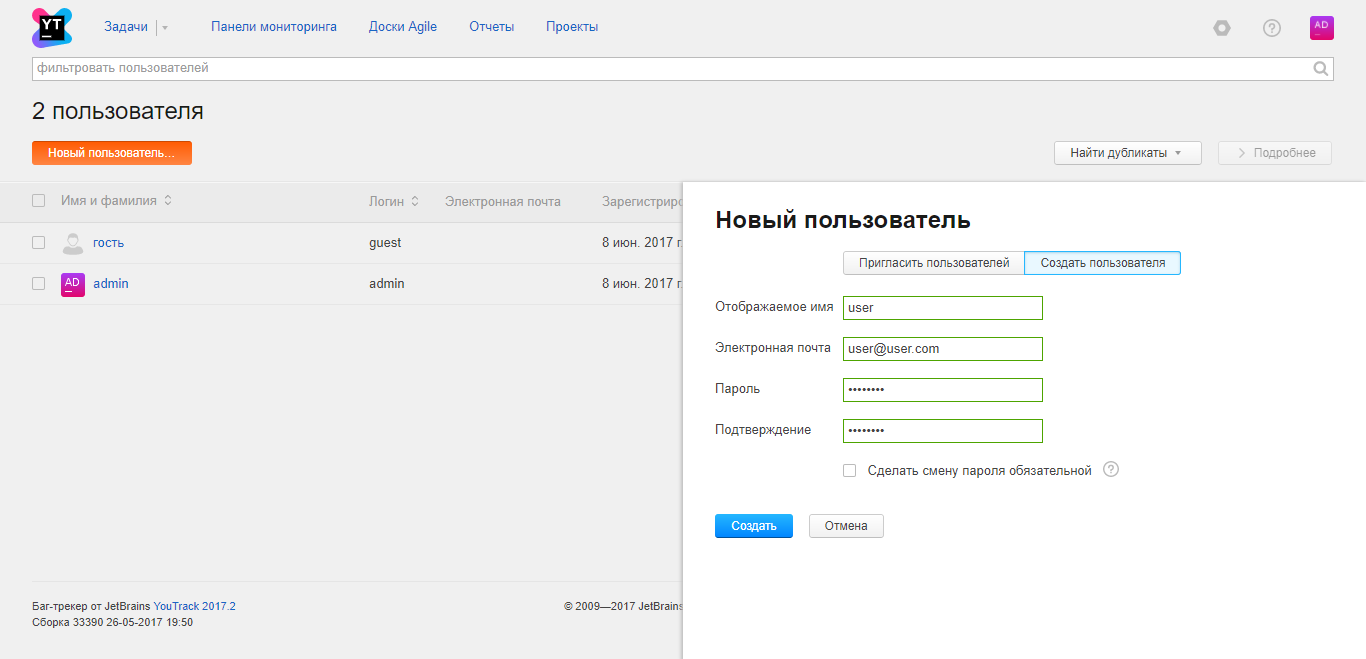


Рисунок 14 – Заполнение формы регистрации

**Создание и настройка проекта в системе YouTrack**

YouTrack – это система управления проектами и система отслеживания ошибок. Она подразумевает под собой создание проекта (Рис. 15) и управление им на всем его жизненном цикле. В YouTrack существует 3 типа проектов:

**Проект по умолчанию**

Проект по умолчанию – это стандартный проект баг-трекера. Он обладает стандартными полями задач и рабочими процессами

**Проект Scrum**

Проект Scrum – это проект баг-трекера, который был настроен для методики Scrum. Он обладает двумя досками Agile. Одна доска служит для разработки, а другая для управления проектом.

**Проект Kanban**

Проект Kanban – это проект баг-трекера, который был настроен для методики Kanban. Он обладает одной доской Agile.

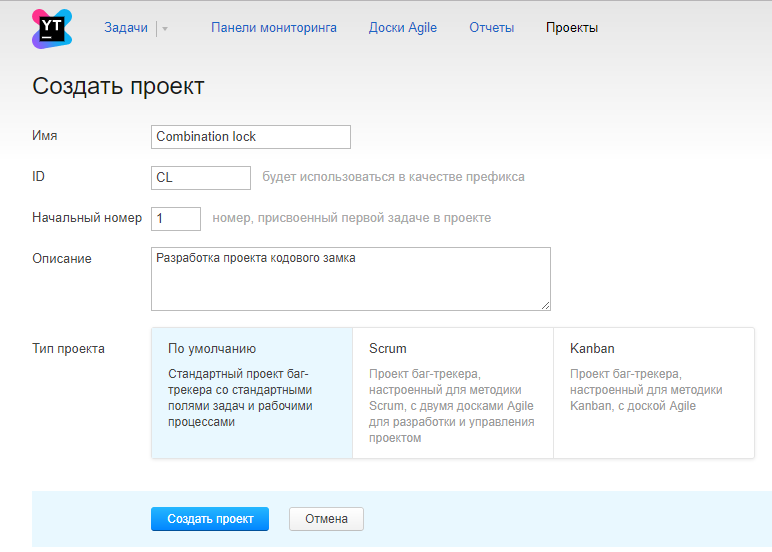


Рисунок 15 – Создание проекта по умолчанию в YouTrack

Чтобы с проектом могли работать несколько пользователей необходимо назначить группу пользователей, которая сможет работать с данным проектом (Рис. 16).

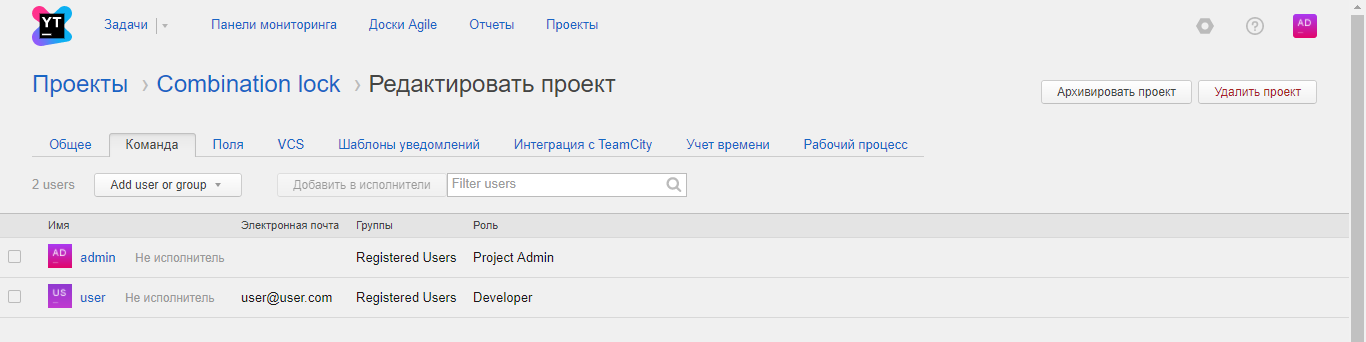


Рисунок 16 – Выбор группы пользователей для работы над проектом

Чтобы в проекте учитывалось время для выполнения задач необходимо включить учёт времени. Также включение учёта времени в проекте необходимо для выполнения разрешения команде регистрировать работу над задачей и создания отчётов по времени.

Для этого в свойствах проекта необходимо перейти на вкладку **Редактировать проект**. Далее необходимо перейти на вкладку **Учёт времени**, поставить галочку рядом с полем **Включить учёт времени** и в **Поле оценки** выбрать значение Estination, а в **Поле «Затраченное время»** - Spent time.

Дополнительно необходимо прикрепить рабочий процесс, который называется jetbrains-youtrack-workTimer (Рис. 17). Для этого в настройках проекта необходимо перейти на вкладку **Рабочий процесс** и прикрепить процесс Work Timer. Чтобы он начал свою работу необходимо создать поля Start timer, stop timer с типом Time.

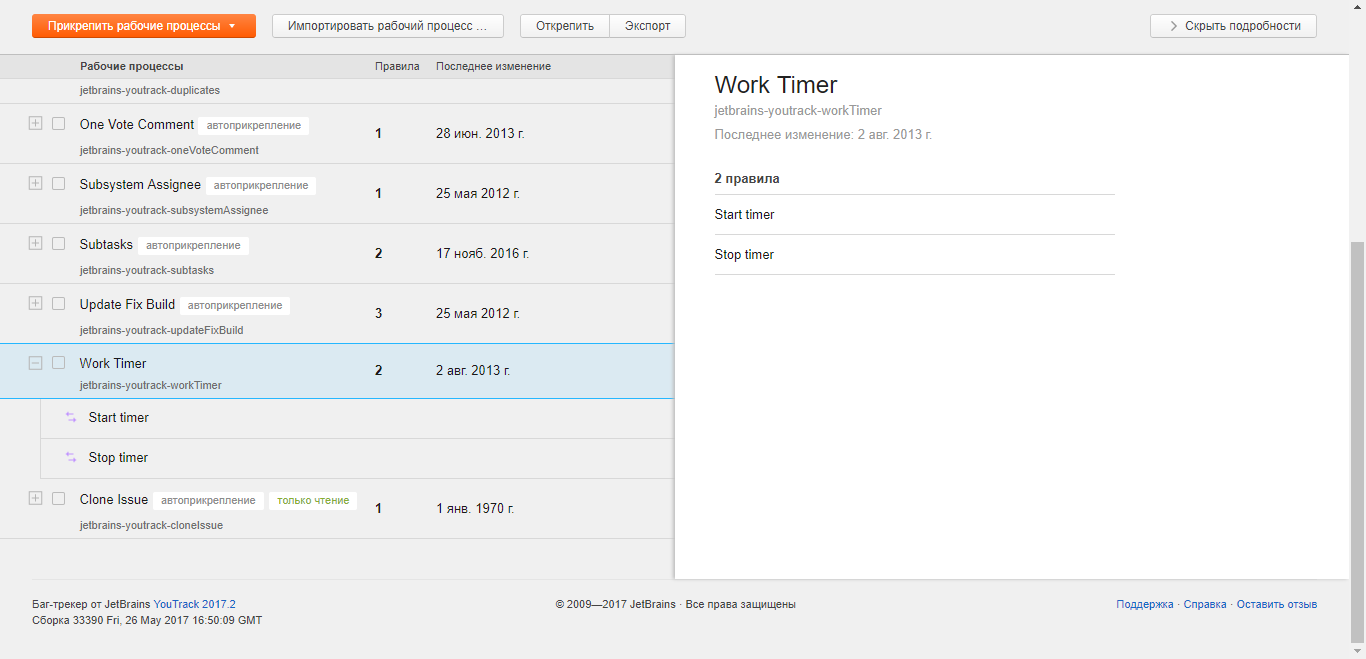


Рисунок 17 – Прикрепление рабочего процесса Work Timer

**Работа с досками в YouTrack**

Управление ходом разработки проекта осуществляется с помощью досок. На доске располагается несколько столбцов, которые подразумевают под собой состояние задачи. Задача может быть в состоянии открыта, в разработке, подлежит обсуждению, исправлена, проверена, развернуто.

Существует 4 типа досок: Доска Scrum, доска Kanban, доска на основе версий, настраиваемая доска. Доска Scrum подразумевает под собой планирование и поэтапную сдачу улучшений в ходе последовательности спринтов. Доска Kanban предполагает визуализацию рабочего процесса в соответствии со спросом и предложением, которые ограничивают текущую работу для каждой команды. На доске, которая основывается на версиях, задачи автоматически присваиваются спринтам путём привязки спринтов к набору значений в настраиваемом поле. Настраиваемая доска предполагает создание доски с нуля для поддержки гибридного подхода Agile.

Чтобы создать и настроить доску необходимо перейти на вкладку **Доски Agile** и нажать на кнопку **Создать**. Из выпадающего списка выбрать доску.

Далее будет рассмотрено создание и настройка доски Scrum (Рис. 18).

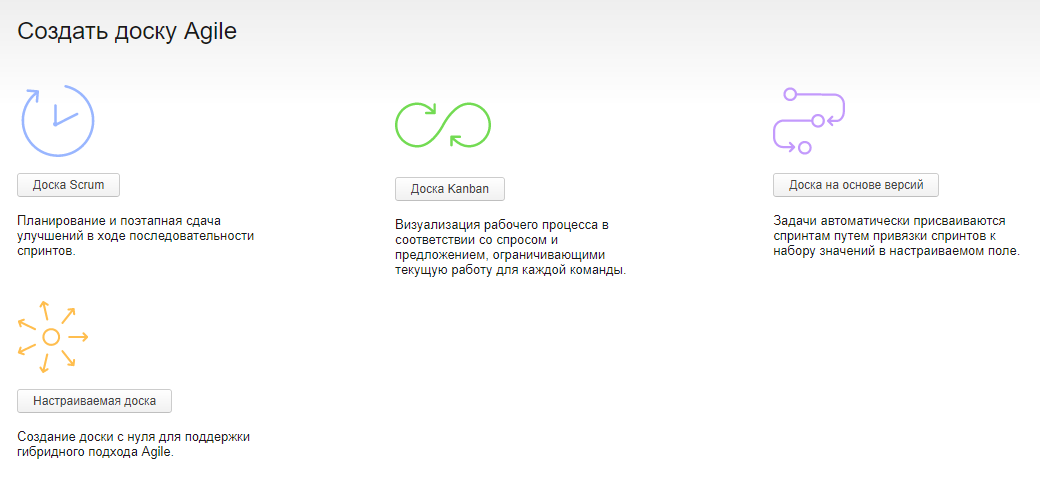


Рисунок 18 – Создание доски Scrum

Далее необходимо заполнить все параметры (Рис. 19) и нажать на кнопку **Создать**.

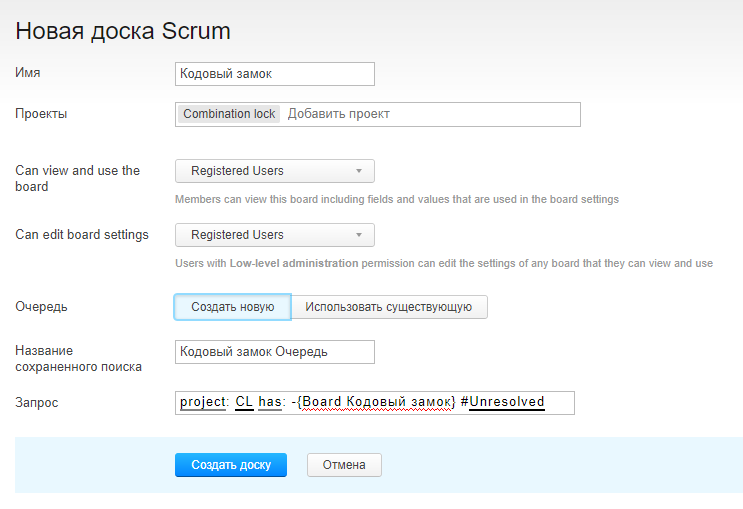


Рисунок 19 – Создание доски Кодовый замок

При создании доски можно производить множество манипуляций с ее компонентами. Можно добавлять и удалять поля, задавать поля, на основе которых будут определяться свимлэйны, задавать Min и Max WIP (минимальное и максимальное количество задач, которые могут находиться в соответствующем столбце доски) (Рис. 20).

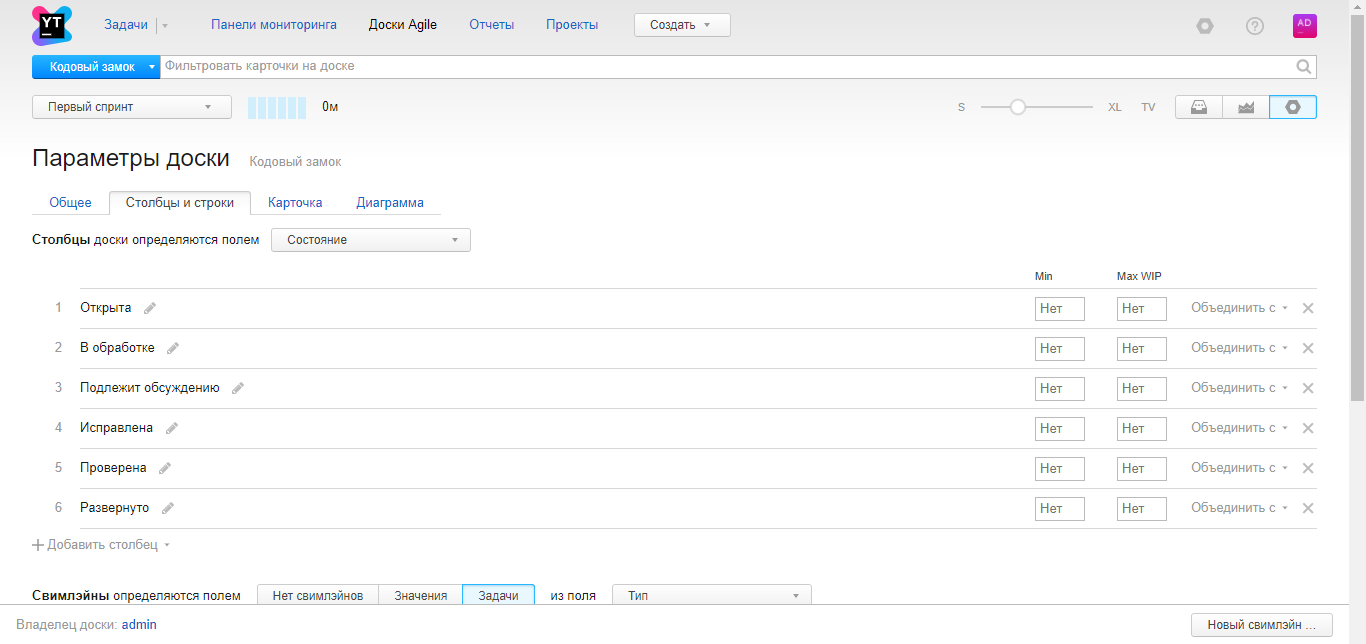


Рисунок 20 – Настройка полей доски

**Создание и манипуляция с задачами и подзадачами в YouTrack**

Одной из основных составляющих в системе управления проектами и системе отслеживания ошибок являются задачи и подзадачи, относящиеся к ним. Пользователи создают задачи при создании проекта. Позже пользователи создают другие задачи во время улучшения проекта. Если возникают какие-то проблемы пользователи тоже создают задачи. Задачи могут находиться в различных состояниях. В одно и то же время задача может находиться только в одном состоянии. Задача имеет свой проект, приоритет, тип, состояние, исполнителя, подсистему, версию, планируемое время выполнения, время, прошедшее с создания задачи. Пользователи могут голосовать за задачи. В топ проходит та задача, за которую проголосовало большинство пользователей. Как правило, это та функция в проекте, которая является очень важной и необходимой. Задачи могут принадлежать к различным доскам.

Для создания задачи (Рис. 21) необходимо нажать на кнопку **Создать задачу** и указать её название, описание, параметры. Также можно создавать и подзадачи. Подзадачи создаются по аналогии с задачами, но есть одно замечание. При создании подзадачи необходимо добавить связь указав в пункте **Добавить связь** родителя.

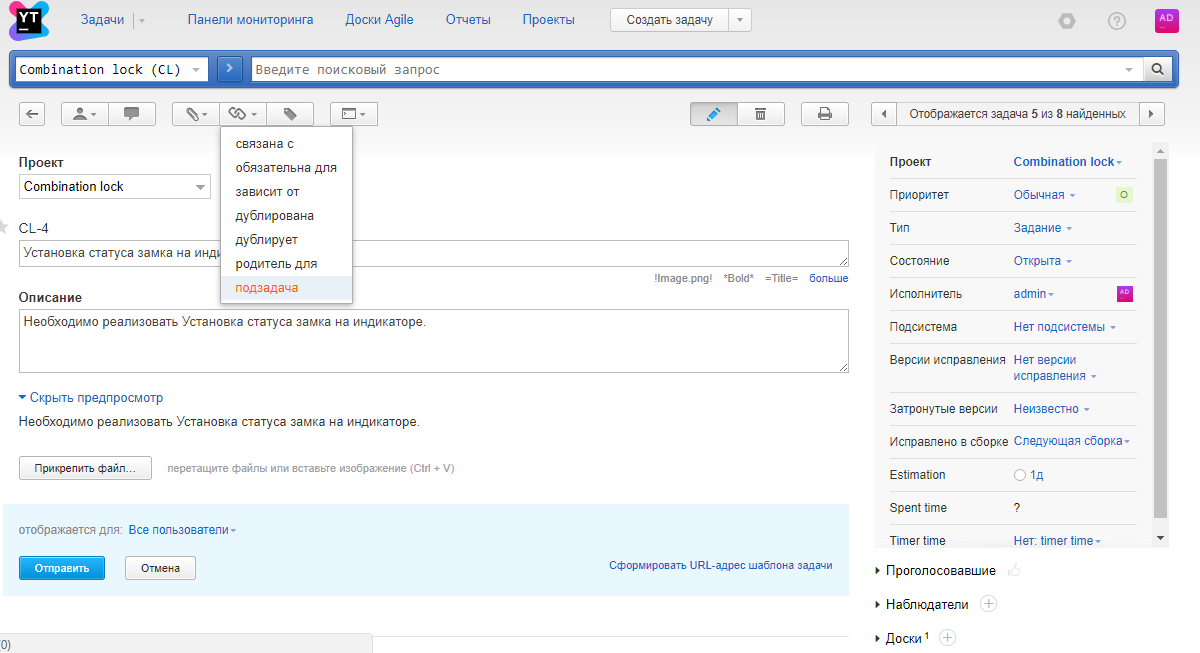


Рисунок 21 – Создание задачи и подзадачи

Чтобы посмотреть список существующих задач, нужно перейти на вкладку **Задачи** (Рис. 22).

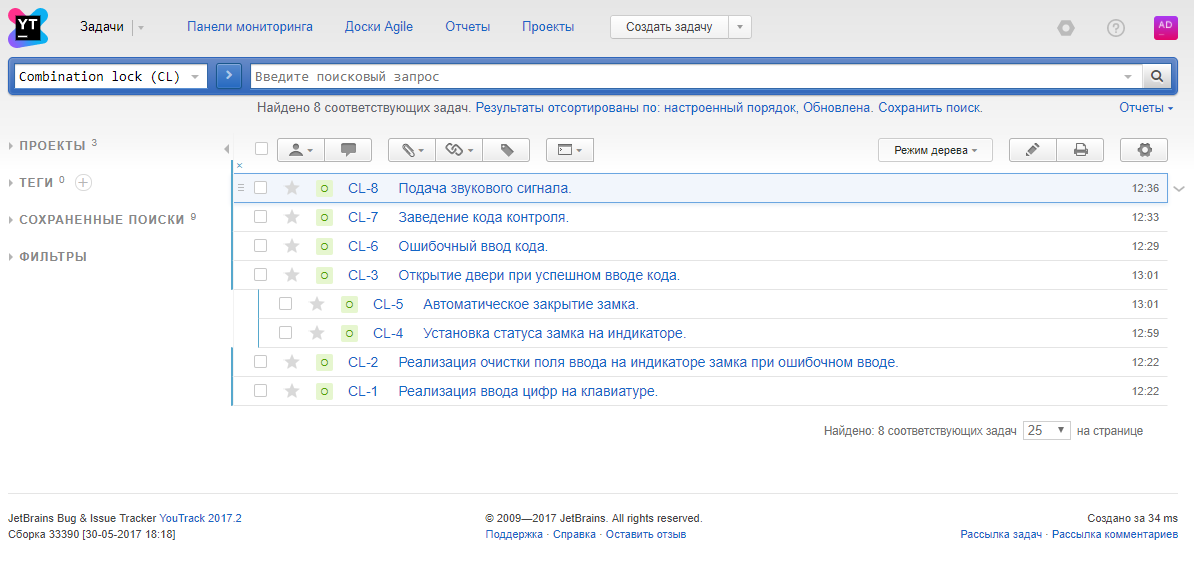


Рисунок 22 – Список задач и подзадач, относящихся к проекту Combination lock (CL).

**Создание панели мониторинга и управление ею**

В YouTrack существует инструмент, который имеет название **Панель мониторинга**. Панель мониторинга может содержать 4 различных типа виджетов: задачи, отчёты, заметки wiki, заметки markdown.

Для создания панели мониторинга необходимо перейти на вкладку **Панели мониторинга** и из выпадающего списка выбрать пункт **New dashboard** (Рис. 23). Чтобы создать виджет (Рис. 24) для задач необходимо нажать на кнопку **Add wiget** и из выпадающего списка выбрать **Issues**. В поле **Type search query** выбрать тип поискового запроса. Например, **#unresolved**. Этот параметр выводит все нерешенные задачи. Чтобы создать виджет для отчётов на панели мониторинга необходимо выбрать **Report** из выпадающего списка и выбрать отчёт, который требуется отобразить на виджете. **Заметки wiki** и **заметки markdown** создаются одинаковым образом.

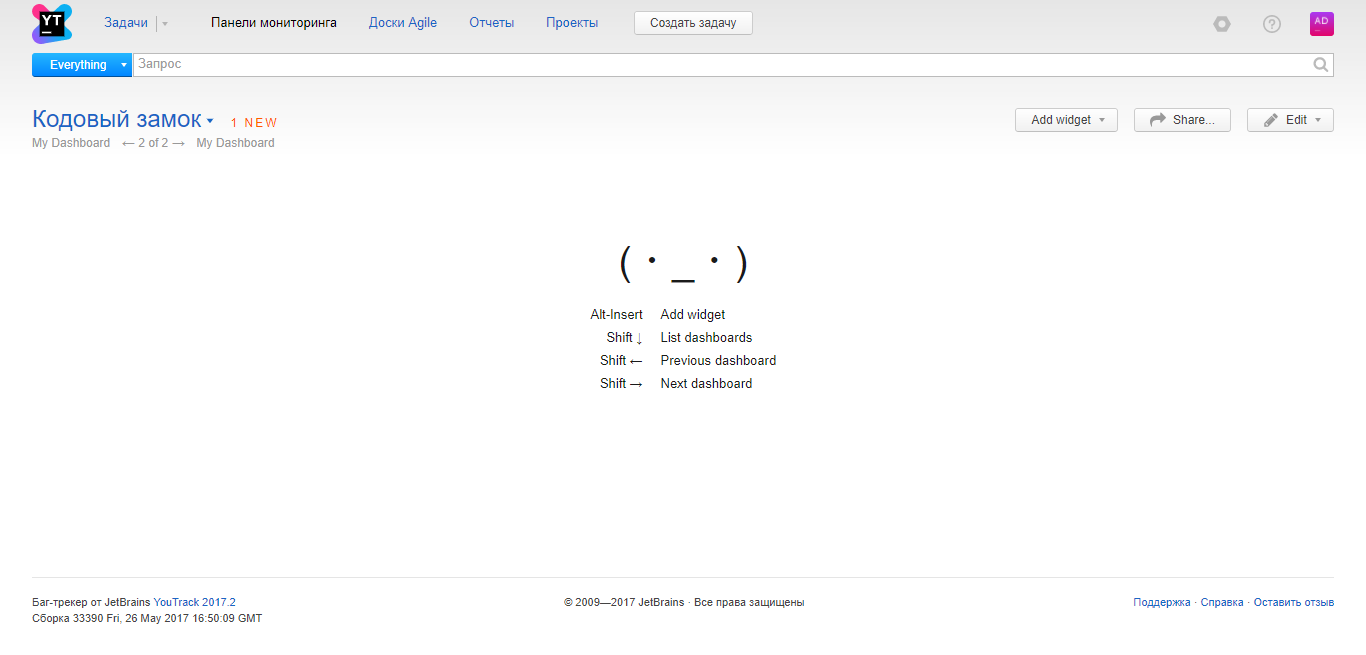


Рисунок 23 – Создание панели мониторинга

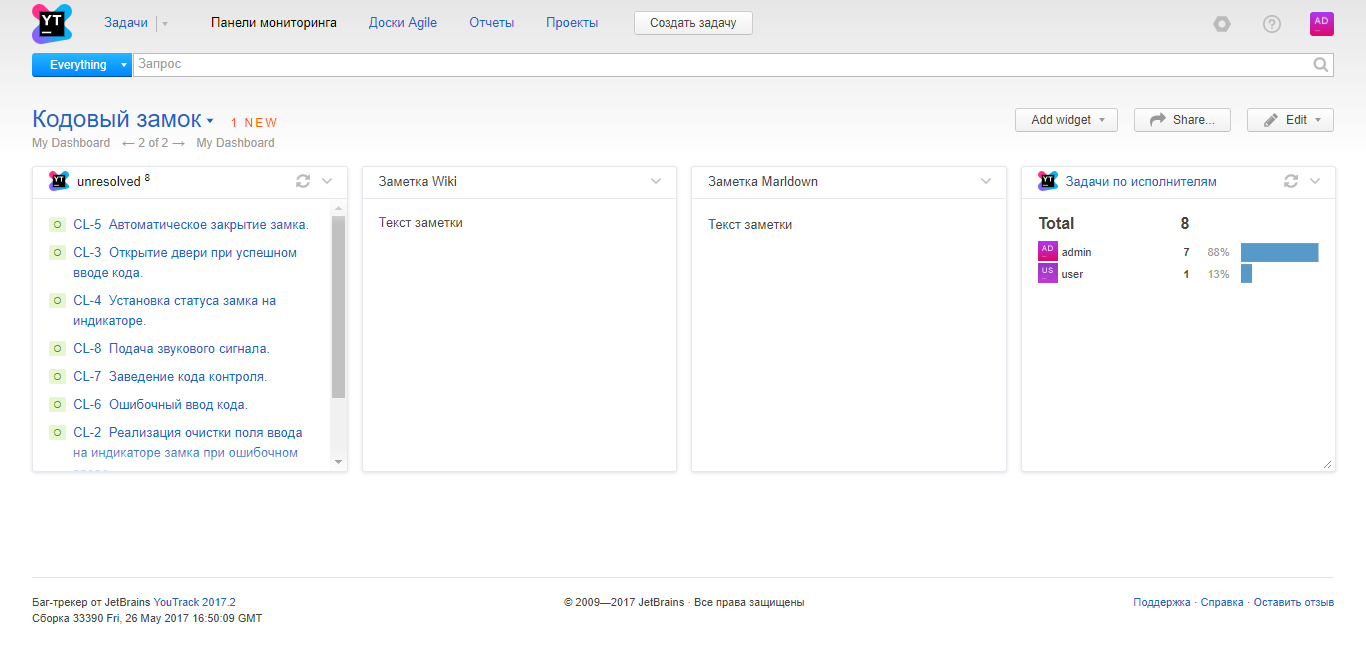


Рисунок 24 – Создание виджетов на панели мониторинга

**Создание отчётов в YouTrack**

В YouTrack существует очень полезный механизм под названием отчёты. Отчёты позволяют визуально отслеживать деятельность всей команды. Также можно отслеживать свои личные результаты. Эти результаты можно отслеживать в реальном времени или за определённый период. С отчётами можно производить различные действия. Их можно экспортировать, печатать, публиковать для других членов команды.

Отчёты делятся на 4 группы: отчёты о распределении задач, отчёты по графикам, отчёты об управлении временем, отчёты о переходе между состояниями.

К примеру, можно построить график о распределении задач по исполнителям (Рис. 25). Для этого необходимо ввести имя отчёта, выбрать проект, задать представление и отображение. После необходимо создать отчёт на панели отображения.

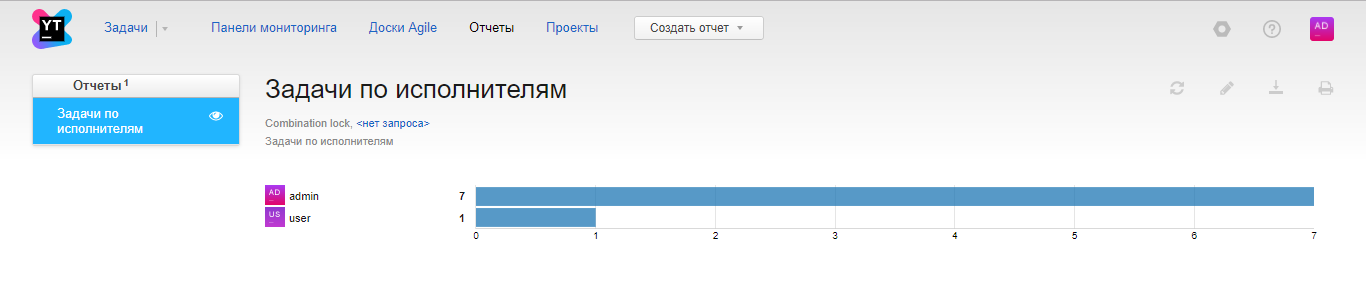


Рисунок 25 – Отчёт: График о распределении задач по исполнителям.

**Работа со строкой поиска**

Одной из достоинств системы YouTrack является наличие большого набора команд для работы с проектами и задачами. Пользоваться YouTrack можно почти не используя клавиатуру оперируя только лишь горячими клавишами на клавиатуре. Чтобы посмотреть комбинации клавиш нужно использовать комбинацию: Ctrl + / (Рис. 26).

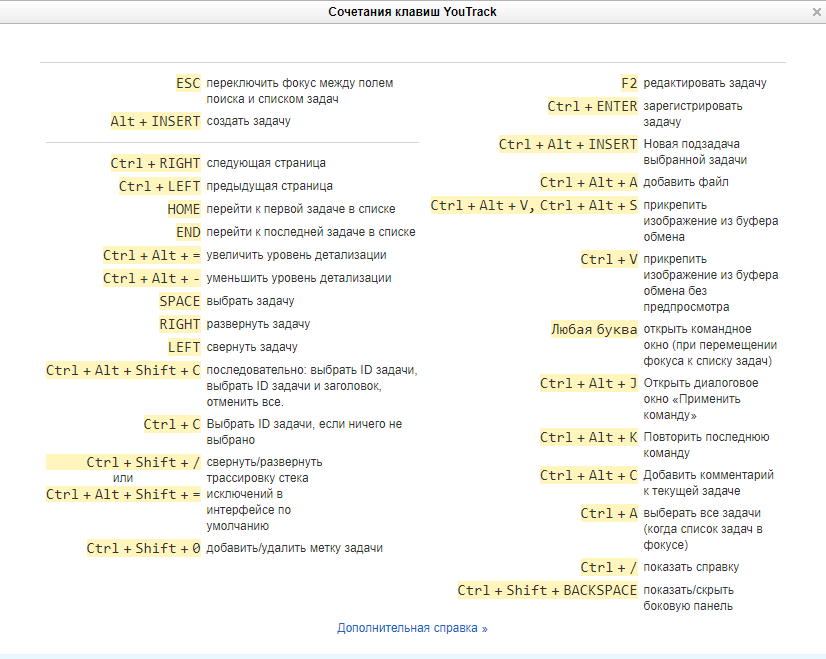


Рисунок 26 – Сочетания клавиш YouTrack

Существует множество команд для работы.

Например:

**проект: CL** – эта команда выводит список задач, которые относятся к проекту с ID CL.

Работу с задачами можно осуществлять с помощью окна **Применить команду**. Оно вызывается комбинацией клавиш Crtl + Alt + J.

Можно изменить состояние задачи набрав в окне **Применить команду**: **В обработке**. Таким образом задача, к которой это действие было применено поменяет статус на **В обработке**.

Задачам можно присваивать теги и искать задачи по тегам. Например, достаточно в окне команд написать: **тег: Замок**. Задаче присвоится тег **Замок**.

Также с помощью окна **Применить команду** можно указать количество времени, которое было затрачено на работу. Команда **работа сегодня 1 ч** указывает задаче, что сегодня над ней работали ровно один час.

В YouTrack существует такой механизм как **Сохранённые поиски**. Он позволяет вводить запрос и сохранять обращение к нему. Таким образом набрав в строке поиска **для: меня #unresolved #Задание сортировать: Приоритет** система выведет нерешённые задачи с типом Задание отсортированные по приоритету, которые были назначены для текущего пользователя (Рис. 27). Это обращение можно сохранить во вкладке **Сохранённые поиски** под определённым именем и обращаться к нему каждый раз, когда это обращение потребуется.

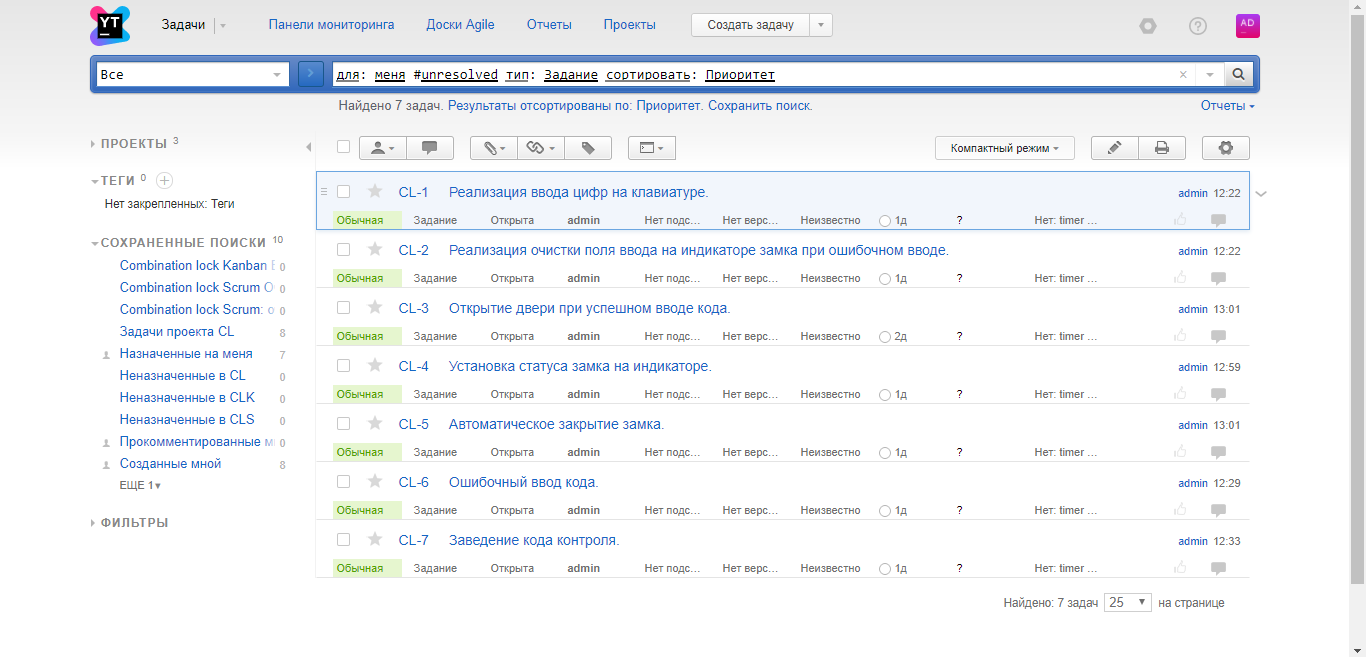


Рисунок 27 – Результаты запроса

YouTrack – это гибкое средство разработки. У команд разработчиков есть широкие возможности, чтобы модернизировать работку в YouTrack под себя. Можно следовать принципам Agile используя Scrum и Kanban в их первозданном виде, а можно использовать гибриды и брать положительные черты и у первой, и у второй методологий. Также в YouTrack существует возможность создавать собственные рабочие процессы и редактировать существующие.

В YouTrack осуществляется интеграция с репозиторием системы контроля версий и с системой непрерывной интеграции TeamCity. Эти функции будут рассмотрены в следующих лабораторных работах.

## 

## **Лабораторная работа №2. Контроль версий**

### **Основные сведения**

Целью данной лабораторной работы является обучить студента работе с системой контроля версий.

Лабораторная работа №2 будет выполняться с использованием Git.

Студент будет разрабатывать проект согласно индивидуальному заданию.

Студент создаст пользователя, создаст локальный репозиторий, осуществит преобразования с локальным репозиторием, добавит удалённый репозиторий, выполнит преобразования с удалённым репозиторием, осуществит интеграцию с GitHub и YouTrack, выполнит преобразования с ветками.

**Общие понятия о системе контроля версий**

**Система управления версиями** – программный продукт, облегчающий работу с меняющейся информацией. Данное ПО может хранить несколько версий проекта, отслеживает кто и когда производил какие-либо действия с проектом. Если возникнет необходимость, то можно будет вернуться к более ранней версии.

**Контроль версий с помощью Git**

**Настройка Git под определённого пользователя**

С помощью команды **git config** можно просмотреть и настроить параметры, которые контролируют всю работу Git. Также можно настроить его внешний вид.

Первым делом после установки Git необходимо указать имя и e-mail (Рис. 28). Это является важным моментом, так как каждый коммит несёт в себе эту информацию. Она включена в коммиты, которые передаёт пользователь. Эта информация не может быть изменена.

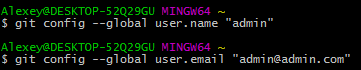


Рисунок 28 – Настройка пользовательских данных

**Проверка настроек**

Чтобы узнать информацию по поводу используемой конфигурации (Рис. 29) можно воспользоваться командой **git config –list**. В ходе выполнения данной команды произойдёт отображение всех настроек.



Рисунок 29 – Вывод информации об используемой конфигурации

**Создание Git-репозитория**

Git-репозиторий можно создать двумя способами.

1. Импортировать в Git уже существующий проект или директорию
2. Клонировать существующий репозиторий с другого сервера

Чтобы начать использовать Git для проекта, который уже существует нужно перейти в директорию проекта и в командной строке ввести **git init** (Рис. 30).

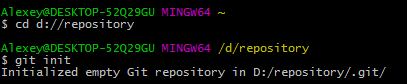


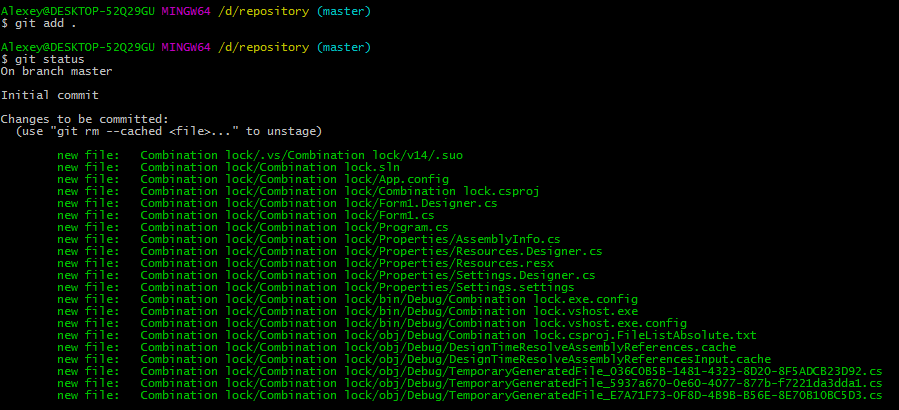
Рисунок 30 – Инициализация Git-директории

Данная команда в текущей директории создаёт поддиректорию, которая называется **.git**. Эта поддиректория включает в себя все файлы репозитория, которые необходимы. Эти файлы представляют из себя основу Git-репозитория. Однако, создание поддиректории **.git**, еще не достаточно, чтобы поместить проект под версионный контроль.

Чтобы добавить файлы под версионный контроль необходимо добавить эти файлы в индекс и произвести первый коммит (Рис. 31). Чтобы осуществить желаемое, необходимо ввести в консоль команду **git add**, несколько раз указав индексируемые файлы. Чтобы добавить все файлы сразу можно ввести команду **git add .**.

После этого необходимо сделать коммит выполнив команду **git commit**.

Команда **git status** производит вывод статуса текущего состояния.



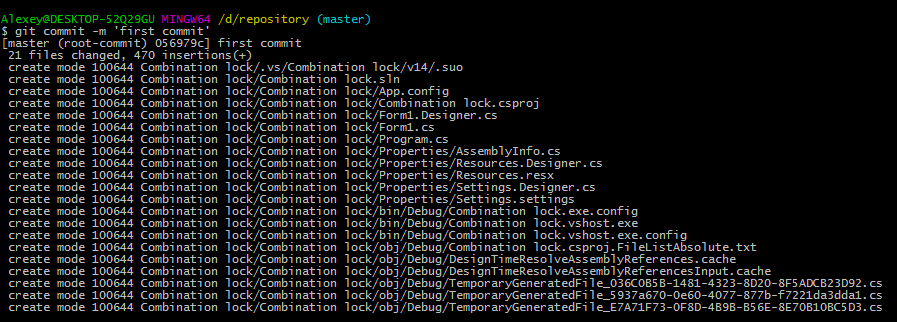


Рисунок 31 – Добавление файлов в индекс, запрос статуса, выполнение коммита изменений

Также есть возможность клонировать существующий репозиторий. Чтобы выполнить это действие необходимо воспользоваться командой **git clone [url]**. Введя в консоль **git clone https://github.com/project/project** будет создана новая директория с именем **project**. В ней будет проинициализирована новая директория с именем **.git**. Также будут скачаны все данные для этого репозитория и будет создана (check out) рабочая копия последней версии.

**Определение состояния файлов**

Существует команда **git status**, которая показывает какие файлы в каком состоянии находятся.

Выполнив команду **git status** (Рис. 32), можно убедиться в том, что в данный момент нет отслеживаемых изменённых файлов. Также не было найдено не отслеживаемых файлов, иначе эти файлы были бы указаны в результате этой команды. Также можно заметить, что команда оповещает пользователя на какой ветке он находится. Ветка **master** – ветка по умолчанию.

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-11_22-47-15.png

Рисунок 32 – Команда **git status** уведомляет пользователя, что изменений обнаружено не было

Добавив новый файл в репозиторий, и выполнив команду **git status** можно обнаружить новый не отслеживаемый файл (Рис. 33).

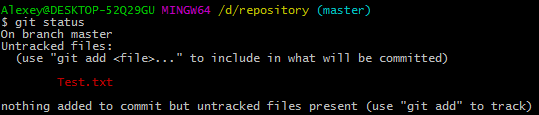


Рисунок 33 – Команда **git status** обнаружила новый не отслеживаемый файл

Нетрудно понять, что так как файл Test.txt находится в секции Untracked, то он является не отслеживаемым. Untracked files говорит о том, что Git обнаружил файл, который отсутствует в предыдущем коммите. Git не будет его добавлять в коммит до тех пор, пока пользователь не даст команду.

**Отслеживание новых файлов**

Команда **git add** используется для того, чтобы осуществить добавление файла под версионный контроль. Для того, чтобы добавить файл Test.txt под версионный контроль необходимо в консоль ввести следующую команду: **git add Text.txt**

Далее необходимо выполнить команду **git status** (Рис. 34)

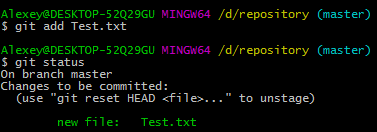


Рисунок 34 – Выполнение команды git status после добавления файла под версионный контроль

Так как файл находится в секции Changes to be commited, то он является проиндексированным. Таким образом, если будет произведён коммит, то версия файлов, которая существовала на момент последнего вызова **git add**, будет помещена в историю снимков состояния.

**Помещение под версионный контроль изменённых файлов**

Если модифицировать файл, который находится под версионным контролем, то при вызове команды **git status** появится уведомление о том, что файл был модифицирован и изменения не были индексированы (Рис. 35).

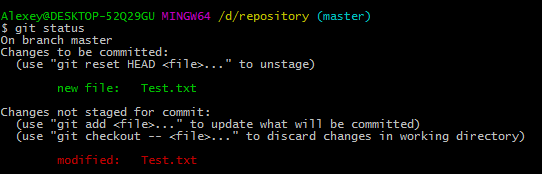


Рисунок 35 – Вызов команды **git status** после изменения файла

Чтобы проиндексировать файл нужно ввести команду **git add**. Команда **git add** является многофункциональной. Её можно задействовать для добавления под версионный контроль новых файлов, для индексации изменений. Кроме того, она используется для указания на те файлы у которых был исправлен конфликт слияния.

Выполнив команду **git add .** и **git status** можно заметить, что изменённый файл был добавлен под версионный контроль (Рис. 36).

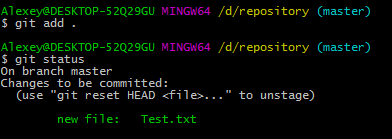


Рисунок 36 – Изменённый файл был добавлен под версионный контроль

**Вывод статуса в кратком виде**

Вывод команды **git status** довольно объёмный. В Git можно вывести сокращённый статус. Таким образом можно отобразить изменения в компактном виде. Введя в консоль **git status –s** или **git statis –short** можно увидеть компактный вывод статуса (Рис. 37).

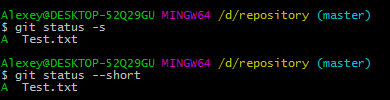


Рисунок 37 – Вывод статуса в компактном виде

**??** – не отслеживаемые файлы

**A** – отслеживаемые файлы

**M** – отредактированные файлы

В пункте M могут быть 2 столбца. Левый – статус файла. Правый – модифицирован ли он после.

**\*M** – модифицирован и не проиндексирован

**MM** – модифицирован, проиндексирован и еще раз модифицирован

**M\*** - модифицирован и проиндексирован

**Игнорирование файлов**

Бывают такие случаи, когда пользователь не хочет автоматически добавлять файлы в репозиторий и видеть файлы в списке не отслеживаемых. Эту проблему решает файл, который называется **.gitignore**. В нём надо указать шаблоны файлов, которые не нужно брать под версионный контроль. Вызвав команду **cat .gitignore** можно увидеть все шаблоны игнорируемых файлов (Рис. 38). В данном примере игнорируются все файлы заканчивающиеся на o и a (Рис. 39).

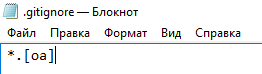


Рисунок 38 – В файле **.gitignore** указаны все шаблоны

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_01-58-41.png

Рисунок 39 – Шаблоны игнорируемых файлов

**Индексированные и неиндексированные изменения**

Если команда **git status** только показывает какие файлы были изменены, то команда **git diff** какие изменения были произведены в файле.

Обычно команда **git diff** даёт ответы на два вопроса: что было изменено, но не проиндексировано, что было проиндексировано и в будущем будет фиксировано. Команда **git diff** какие строки были добавлены и удалены.

Например, файл Test.txt был изменён, но не проиндексирован (Рис. 40).

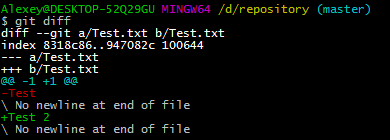


Рисунок 40 – Файл Test.txt был изменён, но не проиндексирован.

Команда **git diff** производит сравнение рабочей директории с индексом. В результате показываются еще не проиндексированные изменения.

Чтобы посмотреть, что было проиндексировано и что будет включено в следующий коммит необходимо ввести в консоль **git diff --staged** (Рис. 41)

--staged и –cached являются синонимами

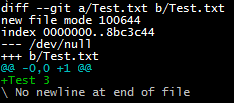


Рисунок 41 – Файл Text.txt был изменён и индексирован.

**Фиксация изменений**

Наступает момент, когда нужно зафиксировать изменения. Если файлы не были проиндексированы, то они не войдут в коммит. Чтобы зафиксировать изменения необходимо воспользоваться командой **git commit**. Если указать ключ **–m**, то можно добавить комментарий. Например, **git commit –m ‘first commit’** (Рис. 42)

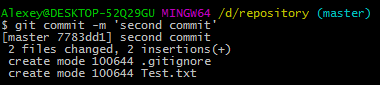


Рисунок 42 – Выполнение коммита

Следует обратить внимание на информацию, которую предоставляет коммит. Как видно из примера коммит выполнен на ветку **master**, контрольная сумма SHA-1 коммита **7783dd1**, было изменено 2 файла, также можно видеть статистику добавленных и удалённых строк в этом коммите. Каждый раз при совершении коммита происходит сохранение снимка текущего состояния проекта.

Порой можно игнорировать индексацию просто указав при выполнении коммита параметр **–a**. Это позволяет выполнить команду **git commit –a**, обойдя команду **git add**. Таким образом происходит автоматическая индексация.

**Удаление файлов**

Чтобы в Git удалить файл нужно удалить его из раздела отслеживаемых файлов, то есть из индекса. После этого необходимо выполнить коммит. Это можно осуществить применив команду **git rm** (Рис. 43). Она удаляет файл из рабочей директории. Если удалить файл из директории вручную, то он будет помечен как не проиндексированный. Удалив его с помощью команды **git rm** можно удалить его из индекса. После следующего коммита файл не будет отслеживаться.

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_02-50-05.png

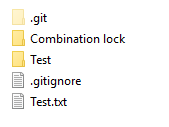
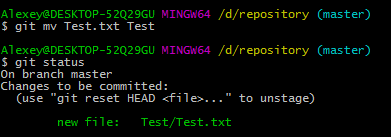
Рисунок 43 – Удаление файла Test.txt из рабочей директории

Если файл был изменён и проиндексирован, то тогда необходимо удалить его принудительно с помощью параметра **–f**. Такой механизм был создан для того, чтобы можно было предотвратить ошибочное удаление данных, если эти данные еще не были закоммичены и их невозможно восстановить из Git.

Чтобы удалить файл из индекса, но оставить его в рабочей директории можно воспользоваться параметром **--cached**. Это может пригодиться если пользователь не занес шаблон в **.gitignore** и ошибочно проиндексировал файл.

**Перемещение файлов**

Чтобы выполнить перемещение файлов в Git необходимо воспользоваться командой **git mv** (Рис. 44).



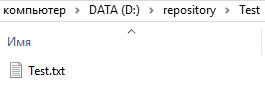
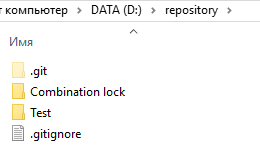


Рисунок 44 – Перемещение файлов

**История коммитов**

После того, как в процессе работы было произведено несколько коммитов скорее всего пользователю потребуется посмотреть какие изменения были произведены. Это можно сделать с помощью команды **git log** (Рис. 45).



Рисунок 45 – Вывод истории коммитов

Таким образом, можно увидеть коммиты, которые были сделаны. Вывод осуществляется в обратном хронологическом порядке.

Если вызвать команду **git log** с параметром **–p**, то можно обнаружить разницу, которая была внесена в коммит. Параметр **-2** показывает, что выведется два коммита. Для того, чтобы увидеть сокращённую статистику для коммита, нужно использовать опцию **–stat** (Рис. 46).

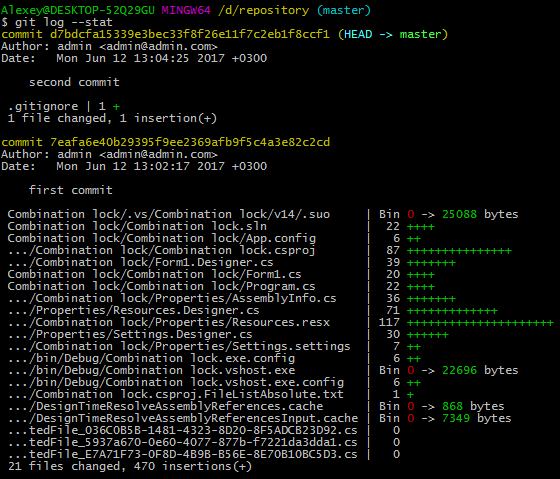


Рисунок 46 – Сокращённая статистика коммитов

Таким образом, можно увидеть список и количество файлов, которые были изменены. Еще эта команда показывает сколько строк было добавлено и удалено в каждом файле.

Команда **–pretty** меняет формат вывода. К примеру значение **oneline** выводит коммит в одну строку (Рис. 47).

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_13-41-47.png

Рисунок 47 – Вывод коммитов в одну строку

Параметр **–format** создаёт собственный формат для вывода информации. Например, **git log –pretty=format**.

Параметр **oneline** и параметр **format** очень полезны при использовании с параметром **–graph**. С помощью параметра **graph** можно построить граф. Этот граф показывает текущую ветку и историю слияний (Рис. 48).

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_13-53-23.png

Рисунок 48 – На данном этапе существует лишь одна ветка, ветка **master**

Существуют параметры ограничения вывода. Например, параметры **–since** и **–until** (Рис. 49).

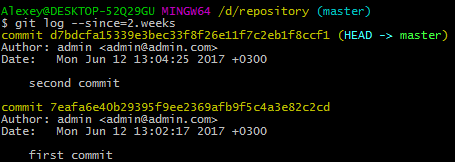


Рисунок 49 – Список коммитов, сделанных за последние две недели

Данная команда взаимодействует с большим количеством разных форматов. Параметр **–author** позволяет искать коммиты по авторам. Параметр **–grep** позволяет искать коммиты по словам в сообщении коммита. Параметр **–all-match** позволяет искать и по авторам и по словам одновременно.

Очень полезным параметром является параметр **–S**. Указав этот параметр и определённую текстовую строку можно найти те коммиты, в которых эта строка была добавлена или удалена.

Например, **git log –Stext**

Выведутся все коммиты в которых на строку Text было оказано какое-либо влияние.

**Операция отмены**

Бывают такие случаи, когда необходимо отменить применённые изменения. Необходимо соблюдать осторожность, так как не все операции отмены в Git являются обратимыми. Отмена действия требуется в том случае, если пользователь выполнил коммит слишком рано. Он мог забыть проиндексировать файл или же добавить комментарий. Параметр коммита **–ammend** может решить эту проблему. Данная команда выполняется чтобы добавить в индекс. В случае неизменности коммита с предыдущего коммита снимок состояния останется прежним. Будет изменён лишь комментарий.

Например, пользователь совершил коммит и забыл проиндексировать файл, который должен был попасть в коммит (Рис. 50).

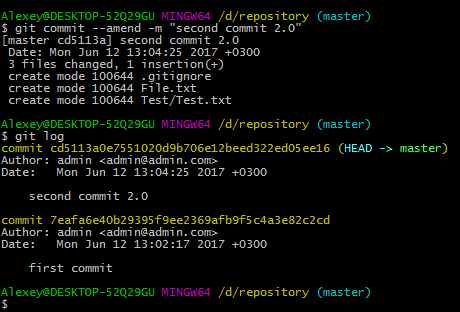


Рисунок 50 – Выполнение операции отмены

**Исключение файла из области подготовки**

Бывает так, что необходимо удалить файл из области подготовки (индекса). Например, пользователь редактировал два файла и оба занёс из в индекс. Позже оказалось, что один из файлов заносить в индекс не стоило. Справка команды **git status** может намекнуть на то, как решить эту проблему. Существует команда **git reset HEAD**, которая отменит добавление файла в индекс (Рис. 51).

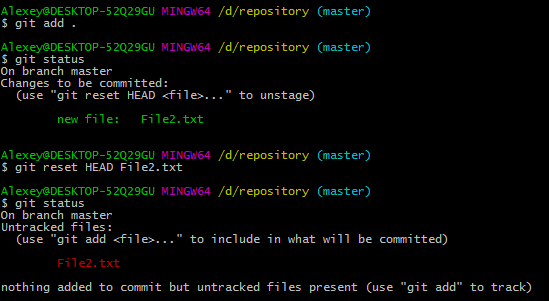


Рисунок 51 – Отмена добавления файла в индекс.

**Отмена изменений для файла, который был изменён**

Бывают ситуации, когда пользователь, изменив файл, не хочет применять эти изменения и вернуть его в прежний вид, который был в прошлом коммите. Команда **git status** даёт подсказку. Чтобы отменить произведённые изменения необходимо воспользоваться командой **git checkout** (Рис. 52).

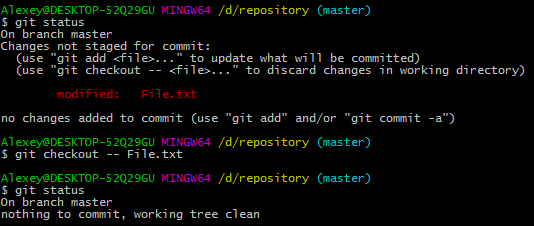


Рисунок 52 – Отмена изменений в файле с помощью команды git checkout

**Осуществление работы с удалёнными репозиториями**

Взаимодействие с удалёнными репозиториями – это важная часть работы с Git. Удалённые репозитории – это версии проекта, которые были сохранены в интернете. Пользователь может обладать несколькими удалёнными репозиториями. Чтобы взаимодействовать с остальными пользователями необходимо уметь работать с удалёнными репозиториями. Можно отправлять данные в удалённые репозитории и получать данные из них. Также можно создавать и удалять удалённые репозитории, осуществлять управление удалёнными ветками.

Используя команду **git remote**, можно посмотреть какие удалённые репозитории были настроены. В том случае, если репозиторий был клонирован, то в консоли появится значения **origin**. Это имя по умолчанию для исходного репозитория. Если указать параметр **–v**, то можно увидеть адреса для чтения и для записи, которые были привязаны к репозиторию.

**Добавление удалённых репозиториев**

Существует веб-сервис для хостинга проектов и их совместной разработки, который называется **GitHub**. **GitHub** предоставляет услуги для создания удалённых репозиториев. На нём можно создать учётную запись и репозитории. Для начала работы с удалёнными репозиториями необходимо создать репозиторий (Рис. 53).

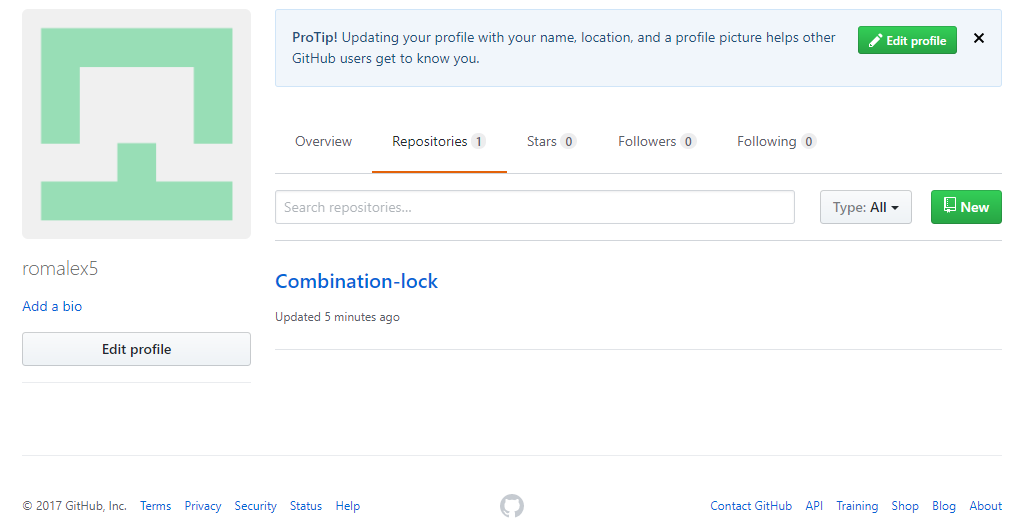


Рисунок 53 – Удалённый репозиторий Combination-lock.

Чтобы добавить удалённый репозиторий необходимо выполнить команду **git remote add [shortname] [url]**. (Рис. 54)

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_16-25-16.png

Рисунок 54 – Добавление удалённого репозитория

**Fetch и pull – получение изменений из удалённого репозитория**

Чтобы получить изменения из репозитория нужно выполнить команду **git fetch [remote-name]**. Эта команда устанавливает связь с удалённым проектом. Далее она забирает те данные проекта, которых еще нет в проекте. После выполнения команды должны появиться все ссылки на все ветки этого удалённого проекта. Эти ветки могут быть просмотрены, также они могут быть слиты.

Команда **git fetch** помещает данные в локальный репозиторий. Но она не производит слияние и модификацию данных. Слияние нужно осуществить вручную.

Команда **git pull** получает данные из удалённой ветки и автоматически производит слияние. Таким образом, команда **git pull** выполняет действие сразу двух команд, **fetch** и **merge**.

**Добавление данных в удалённый репозиторий**

Чтобы отправить данные в удалённый репозиторий нужно воспользоваться командой **git push [remote-name] [branch-name]** (Рис. 55).

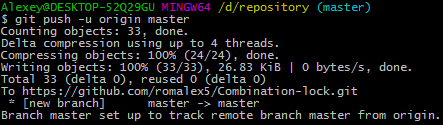


Рисунок 55 – Добавление изменений в удалённый репозиторий

Таким образом, производится отправка ветки **master** на сервер **origin**.

**Данные об удалённом репозитории**

Чтобы получить данные о репозитории необходимо выполнить команду **git remote show [remote-name]** (Рис. 56).

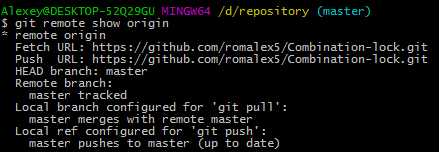


Рисунок 56 – Данные о репозитории

Данная команда выдаёт URL удалённого репозитория. Также она выдаёт данные об отслеживаемых ветках. Например, команда сообщает, что если будет выполнена команда **git pull**, то ветка **master**, находящаяся на сервере, будет слита с текущей веткой.

**Смена имени удалённого репозитория**

Смену имени ссылки удалённого репозитория можно осуществить с помощью команды **git remote rename** (Рис. 57). Также поменяются названия удалённых веток.

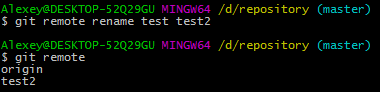


Рисунок 57 – Смена имени ссылки удалённого репозитория

**Удаление удалённого репозитория**

Чтобы удалить ссылку на удалённый репозиторий можно воспользоваться командой **git remote rm** (Рис. 58).

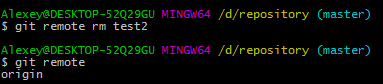


Рисунок 58 – Удаление ссылки на удалённый репозиторий

**Метки в Git**

В Git существует возможность отмечать моменты в истории разработки как важные. Обычно метки используются для отметки версий выпусков. Чтобы узнать какие метки существуют нужно воспользоваться командой **git tag**. Найти нужную метку можно с помощью параметра **–l**.

В Git существует два типа меток. **Легковесные** и **аннотированные**.

**Легковесная метка** – указатель на коммит.

**Аннотированная метка** – хранится в базе данных Git как объект. Аннотированные метки обладают контрольной суммой. В аннотированной метке содержатся: имя пользователя, который поставил метку, e-mail, дату, комментарий. Чтобы создать аннотированную метку необходимо воспользоваться командой **git tag** с параметром **–a** (Рис. 59).

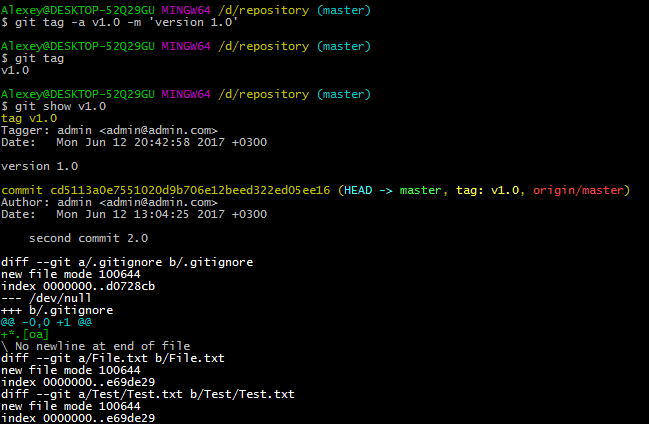


Рисунок 59 – Создание метки **git tag v1.0**

Команда **git show v1.0** отобразит информацию об авторе метки, дате отметки коммита и сообщение.

**Легковесная метка** – это контрольная сумма коммита, которая была сохранена в файл. Чтобы создать легковесную метку достаточно лишь не передавать параметры: **-a**, **-s**, **-m** (Рис. 60).

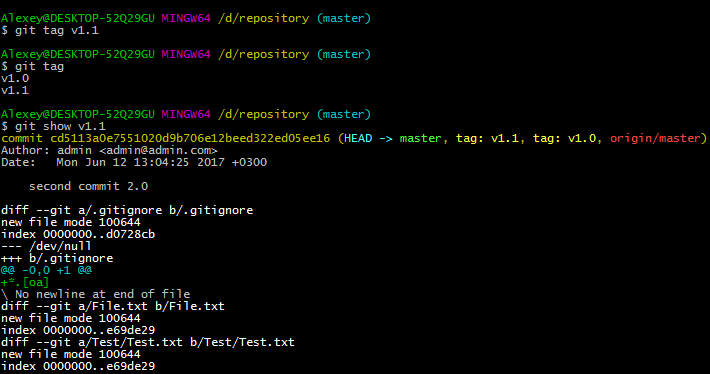
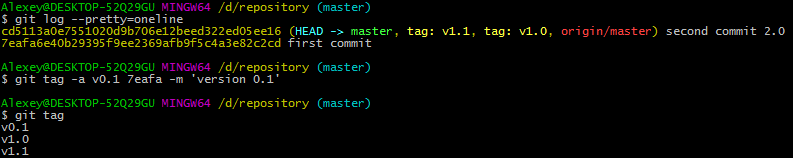


Рисунок 60 – Создание легковесной метки

При выполнении команды **git show**, дополнительная информация отображаться не будет.

**Отметка пройденного коммита**

Существует возможность ставить метки на коммиты, которые уже были пройдены. Чтобы добавить метку на пройденный коммит, нужно указать его контрольную сумму или её часть в команду (Рис. 61).



C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-12_21-03-13.png

Рисунок 61 – Отметка уже пройденного коммита.

По умолчанию метки в репозиторий не добавляются. Чтобы добавить метку в репозиторий необходимо воспользоваться командой **git push origin [имя метки]** (Рис. 62).

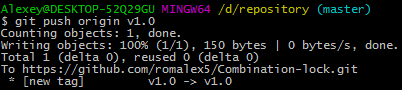


Рисунок 62 – Добавление метки в репозиторий

Существует возможность добавить все метки в репозиторий. Для этого нужно запустить команду **git push origin –tags** (Рис. 63).

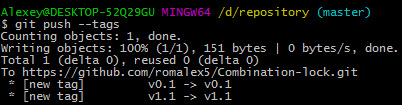


Рисунок 63 – Добавление всех меток в репозиторий

**Интеграция GitHub с YouTrack**

В репозитории GitHub есть возможность осуществить интеграцию с YouTrack. Настройка интеграции проекта YouTrack с репозиторием GitHub позволит обновлять задачи из комментария к коммиту во время работы с VCS. Также, есть возможность получать доступ к коммитам непосредственно из задачи в YouTrack. Для этого необходимо перейти на вкладку **Администрирование** и выбрать пункт **Интеграция с репозиториями** (Рис. 64).

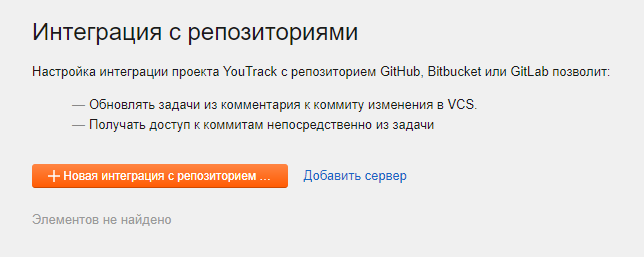


Рисунок 64 – Осуществление интеграции с репозиторием

После этого нужно заполнить форму. Указать сервер, создать Oauth токен. Для создания токена пользователь будет перенаправлен на сайт GitHub. После указания имени и создания токена, будет сгенерирован сам токен. Его необходимо скопировать и вставить в поле Токен OAuth в YouTrack (Рис. 65).

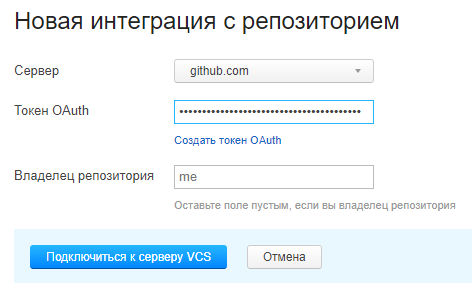


Рисунок 65 – Заполнение формы

После этого надо подключиться к серверу VCS, чтобы удостовериться, что интеграция работает правильно.

Далее следует выбрать проект в YouTrack к которому необходимо подключиться, выбрать репозиторий из GitHub, указать коммитеров, отслеживаемые ветви, группы пользователей, которые могут работать с этим проектом.

Во время выполнения следующего коммита, в комментарии необходимо указать задачу из проекта в YouTrack. Например, **Added new function #CL-1 Fixed**. Статус задачи в YouTrack изменится. Задача будет иметь статус **Исправлена**.

**Ветвление**

В системах контроля версий существует механизм, который называется ветвление. С помощью ветвления можно отклоняться от основной линии разработки и выполнять работу, не затрагивая основную линию разработки. Позже можно произвести слияние с основной ветвью разработки и продолжить работу дальше.

Основной веткой в Git является ветка **master** (её можно переименовать). Когда происходит коммит, то пользователь получает основную ветку, которая указывает на последний коммит. Когда выполняется новый коммит, указатель перемещается вперед.

**Создание ветки**

При создании новой ветки осуществляется создание нового указателя, который необходим для дальнейшего перемещения. Чтобы создать новую ветку необходимо воспользоваться командой **git branch** (Рис. 66).

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-13_20-06-54.png

Рисунок 66 – Создание новой ветки

Эта команда создаёт новый указатель на коммит, на который указывает текущая ветка.

Для того, чтобы понять в какой ветке происходит работа в текущий момент существует указатель **HEAD**. Это указатель, который указывает на локальную ветку, в которой производится работа в данный момент.

Чтобы понять, куда указывают указатели веток, можно воспользоваться командой **git log, указав параметр –decorate** (Рис. 67).

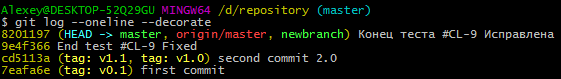


Рисунок 67 – Вывод информации об указателях веток

**Переключение ветки**

Для того, чтобы переключаться между ветками необходимо воспользоваться командой **git checkout** (Рис. 68) После этого указатель **HEAD** переключится на ветку **newbranch**

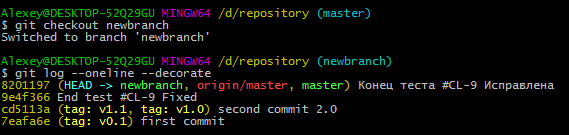


Рисунок 68 – Демонстрация переключения между ветками

Переключив ветку и сделав новый коммит, работа продолжится в ветке **newbranch** (Рис. 69).

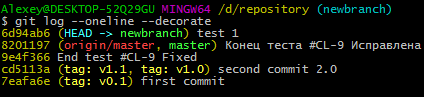


Рисунок 69 – Указатель на ветку HEAD был перемещён вперёд после коммита.

Таким образом, указатель на ветку **newbranch** был перемещён вперед, а указатель на ветку **master** указывает на коммит, во время которого была выполнена команда **git checkout**.

Произведено переключение на ветку master (Рис. 70).

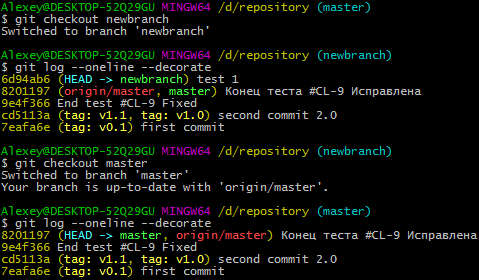


Рисунок 70 – Переключение на ветку **master**

Таким образом, было произведено переключение указателя **HEAD** на ветку **master**, а также был произведён возврат файлов в рабочей директории в состояние, которое было сохранено в снимке, на который указывает ветка. Таким образом, теперь все изменения, которые будут произведены, будут отнесены к старой версии разрабатываемого проекта.

Для наглядности будет произведён коммит. На данном этапе история проекта пошла по двум путям. Была создана ветка, было произведено переключение на неё, затем была произведена работа с ней. Позже был произведён возврат в основную ветку и снова была произведена работа в ней. Эти изменения независимы друг от друга. Можно переключаться между ними. Позже, при желании, можно слить их вместе. Разветвлённую историю можно посмотреть при помощи команды **git log** (Рис. 71). Эта команда продемонстрирует историю коммитов, а также покажет где в данный момент находятся указатели веток и каким образом велась история проекта.

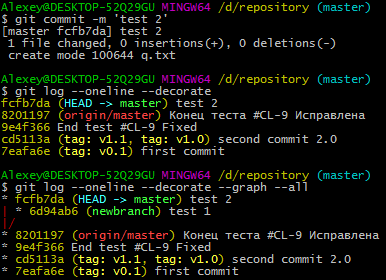


Рисунок 71 – Разветвлённая история

**Основы ветвления и слияния**

Для того, чтобы создать ветку и сразу переключиться на неё необходимо выполнить команду **git checkout** с параметром **–b** (Рис. 72).

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-18_16-46-20.png

Рисунок 72 – Создание и автоматическое переключение на ветку

Таким образом, создаётся ветка **testbranch** и указатель **HEAD** автоматически перемещается на неё (Рис. 73).

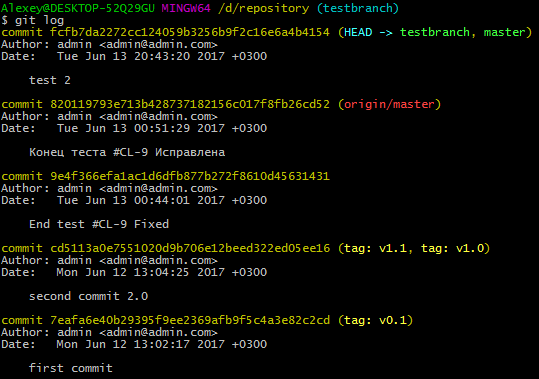


Рисунок 73 – Создание ветки **testbranch** и автоматическое переключение указателя **HEAD**

Далее необходимо произвести коммит (Рис. 74).

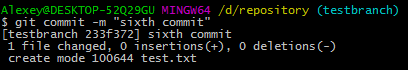


Рисунок 74 – Выполнение коммита

Если у пользователя возникла необходимость перейти на ветку **master** с ветки **testbranch**, но в ветке **testbranch** находятся изменения, которые не попали в последний коммит или конфликтуют с веткой **master**, то Git не даст совершить переключение.

Чтобы выполнить слияние веток, нужно воспользоваться командой **git merge** (Рис. 75).

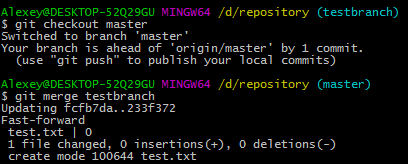


Рисунок 75 – Слияние веток **master** и **testbranch**

Таким образом, было произведено переключение на ветку, в которую необходимо включить изменение. А также, было произведено слияние ветки.

Чтобы удалить ветку, которая стала не нужна, необходимо выполнить команду **git branch –d** (Рис. 76).

C:\Users\Alexey\YandexDisk\Скриншоты\2017-06-18_17-14-44.png

Рисунок 76 – Удаление ветки **testbranch**

## 

## **Лабораторная работа №3. Непрерывная интеграция**

### **Основные сведения**

Целью данной лабораторной работы является обучить студента работе с системой непрерывной интеграции.

Лабораторная работа №3 будет выполняться с использованием TeamCity.

Студент будет разрабатывать проект согласно индивидуальному заданию.

Студент создаст проект из GitHub, создаст и настроит проект вручную, создаст конфигурацию сборки, настроит систему контроля версий, настроит автоматический запуск задания, настроит компиляцию проекта, выполнит сборку в автоматическом режиме, выполнит проверку TeamCity на отслеживание ошибочных сборок.

**Общие понятия о системе непрерывной интеграции**

Одной из практик разработки ПО является непрерывная интеграция. При непрерывной интеграции происходит слияние рабочих копий в общую основную ветвь разработки. Слияние может происходить по нескольку раз в день.

Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции за счёт более раннего обнаружения ошибок и устранения противоречий. Также непрерывная интеграция используется для скорейшего выявления и решения интеграционных проблем. Это реализуется путем частых автоматизированных сборок проекта.

Работа TeamCity выглядит следующим образом. Если возникают изменения в ветке репозитория, то происходит выполнение сценария, который может включать – сборку приложения, прогон тестов, выполнение других сценариев, заливку файлов на удалённый сервер и многое другое.

Чтобы запустить процесс сборки используется программа-агент. Она принимает команды от TeamCity-сервера. Эта программа запускается на любой из основных операционных систем. На одном компьютере может быть несколько агентов. Они могут запускаться параллельно. Однако, один агент одновременно может работать только с одним проектом. Во время старта задачи TeamCity выбирает первый незанятый агент.

**Непрерывная интеграция с помощью TeamCity**

Взаимодействие с TeamCity происходит через браузер. Для того чтобы начать работу с TeamCity необходимо перейти по адресу, настроенному в процессе установки.

Например: 127.0.0.1:82 или localhost:82

TeamCity обладает дружелюбным пользовательским интерфейсом (Рис. 77). В нем всё находится под рукой.

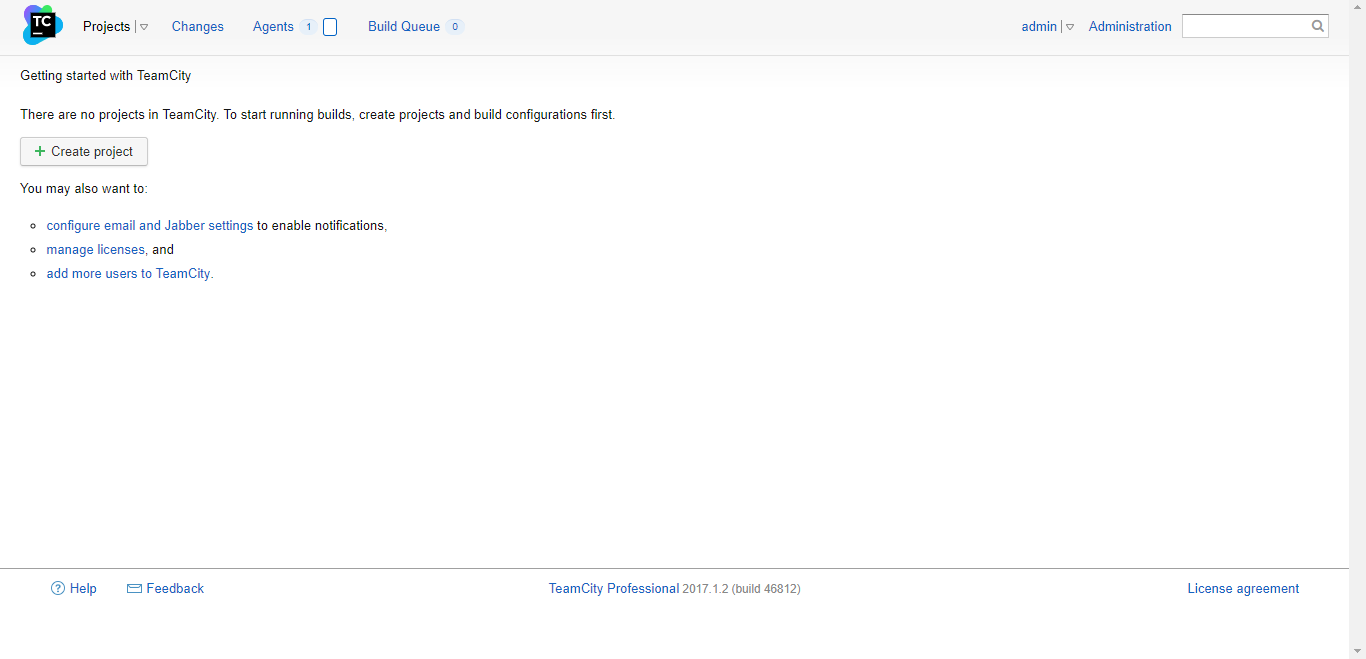


Рисунок 77 – Интерфейс TeamCity

**Создание проекта в TeamCity**

Для того, чтобы начать работу в TeamCity, необходимо создать проект. В TeamCity проект можно создать несколькими способами. Вручную, из URL-адреса репозитория, из GitHub, из Bitbucket Cloud, из Visual Studio Team Services.

**Создание проекта из GitHub**

Если создавать проект из GitHub, то потребуется зарегистрировать приложение TeamCity в GitHub (Рис. 78).

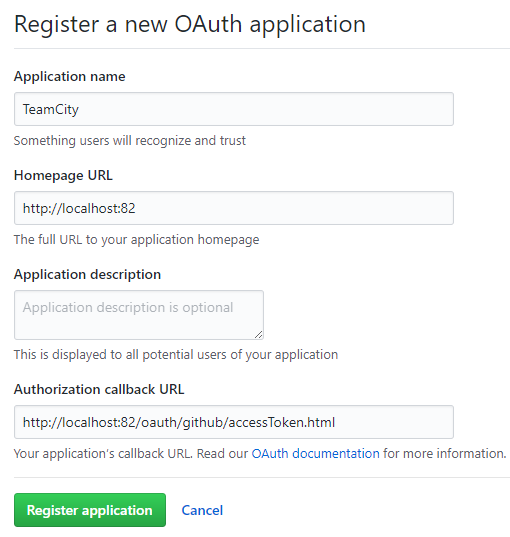


Рисунок 78 – Регистрация TeamCity на GitHub

Таким образом, осуществится связь GitHub и TeamCity путём передачи в GitHub данных из TeamCity о Homespace URL и Callback URL. После регистрации TeamCity в GitHub, GitHub предоставит данные Client ID и Client secret. Их необходимо указать в форме добавления соединения в TeamCity (Рис. 79).

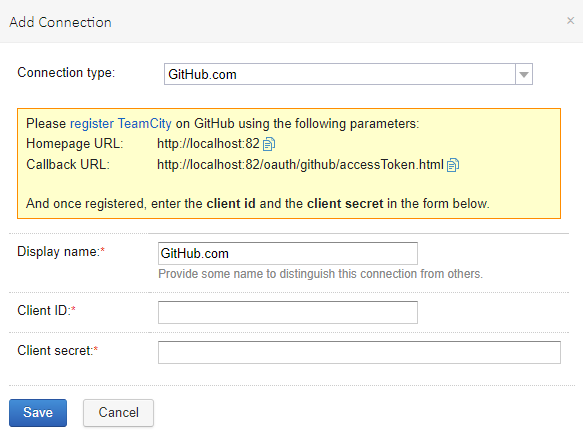


Рисунок 79 – Добавление соединения

После этих действий необходимо войти в GitHub через форму создания проекта в TeamCity и выбрать нужный репозиторий (Рис. 80).

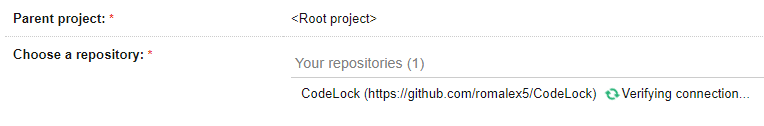


Рисунок 80 – Выбор репозитория

Далее нужно окончательно подтвердить создание проекта (Рис. 81).

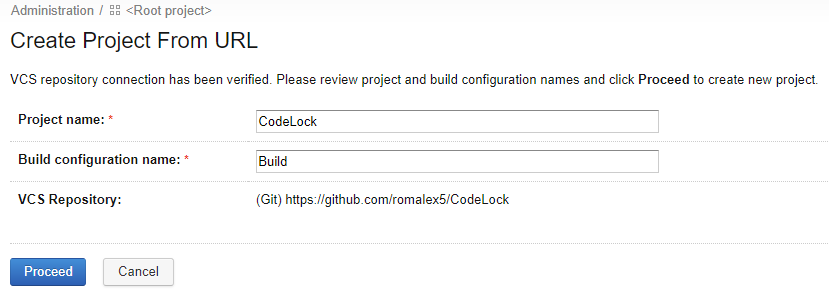


Рисунок 81 – Завершение создания проекта

Таким образом, при создании проекта из GitHub, произойдет автоматическая настройка конфигурации сборки. Будут автоматически настроены: настройки контроля версий, шаги сборки, условия сборки (Рис. 82).

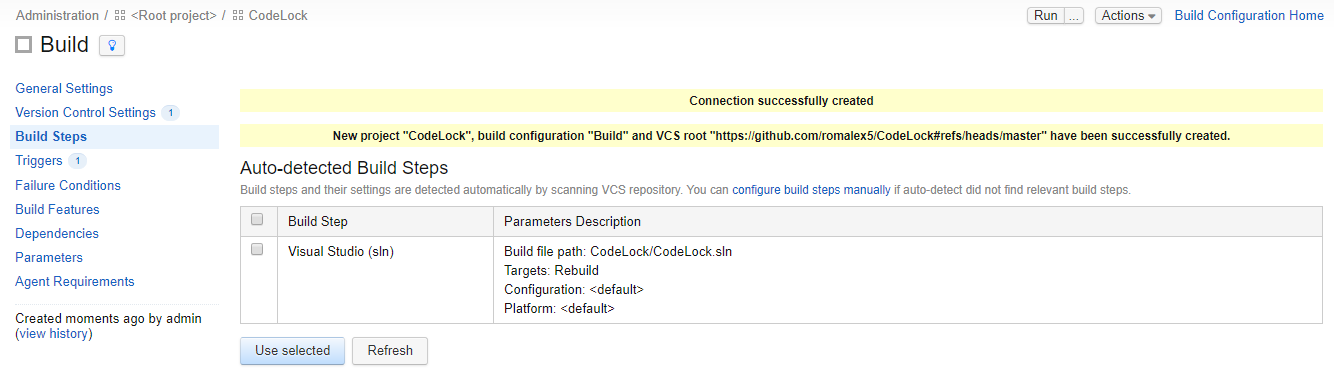


Рисунок 82 – Автоматическая настройка сборки

**Создание проекта вручную**

Также проект можно создать вручную (Рис. 83). Для этого необходимо заполнить форму создания проекта.

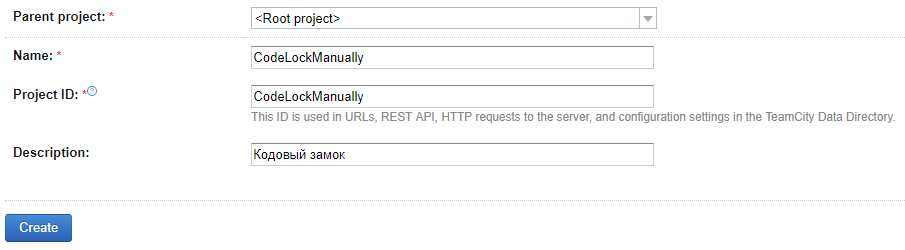


Рисунок 83 – Создание проекта

Далее нужно выполнить создание конфигурации сборки (Рис. 84). Она определяет сценарий сборки.

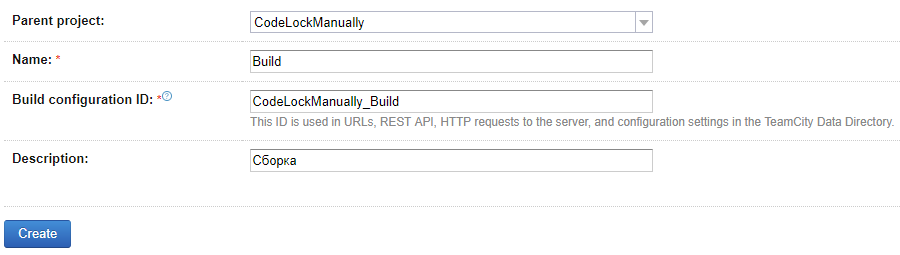


Рисунок 84 – Создание конфигурации сборки

После создания конфигурации сборки её необходимо настроить. Первым делом следует настроить используемую систему контроля версий (Рис. 85).

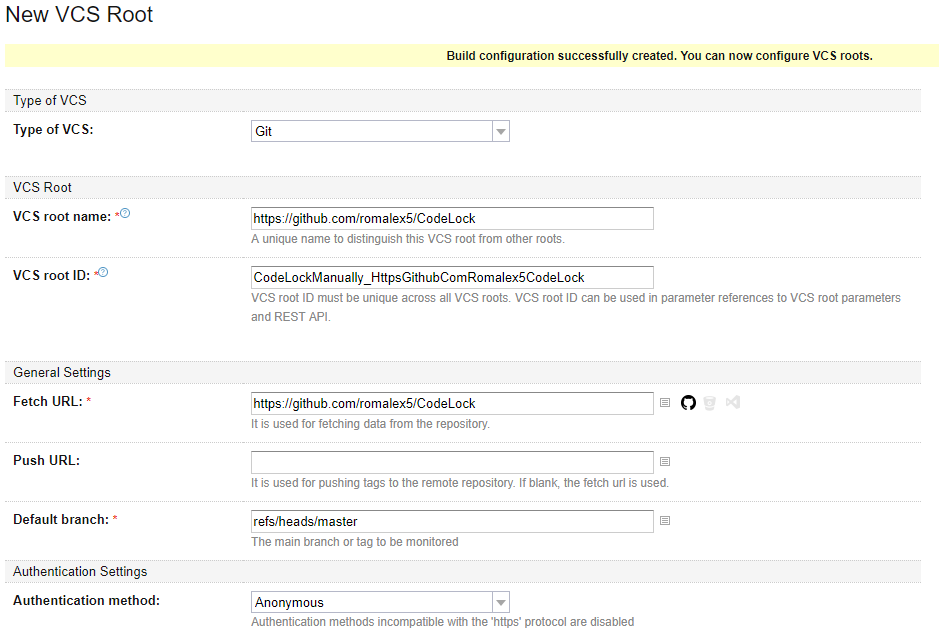


Рисунок 85 – Настройка системы контроля версий

Заполнив все данные нужно проверить соединение с репозиторием (Рис. 86).

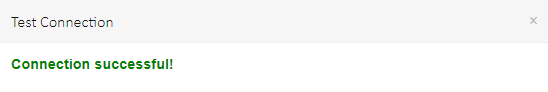


Рисунок 86 – Проверка соединения

После настройки VCS необходимо настроить автоматический запуск задания. Для этого в условиях сборки необходимо выбрать значение VCS Trigger (Рис. 87). Если будут использованы настройки по умолчанию, то он раз в минуту будет проверять наличие новых коммитов в репозитории. Если он обнаружит новые коммиты, то задания будут запущены на выполнение.

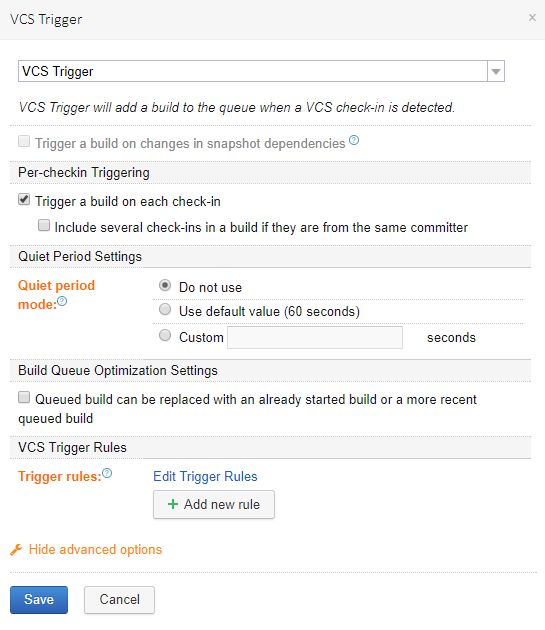


Рисунок 87 – Настройка автоматического запуска задания

Далее необходимо произвести компиляцию проекта. В шагах сборки необходимо выбрать runner type (Рис. 88). В качестве runner type будет использоваться Visual Studio (sln).

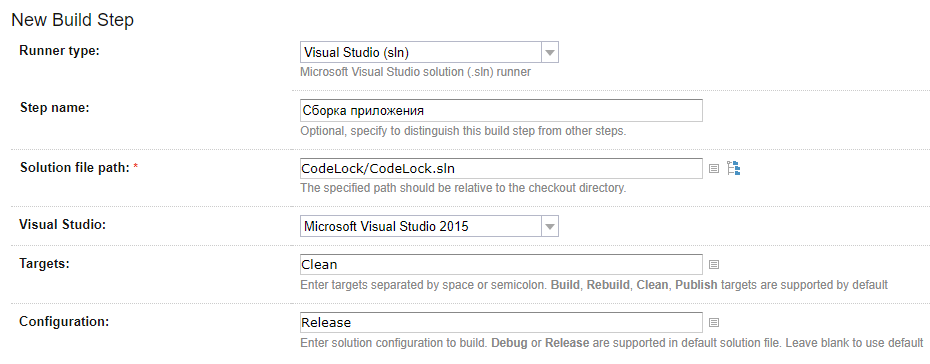


Рисунок 88 – Настройка компиляции проекта

**Запуск проекта на сборку**

Для того чтобы запустить проект на сборку необходимо нажать на кнопку **Run** (Рис. 89).

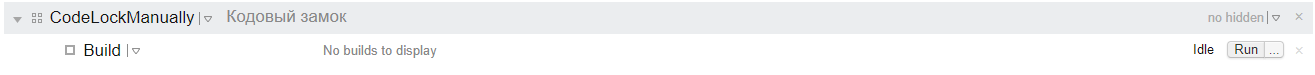


Рисунок 89 – Новый проект и новая конфигурация сборки

В случае благоприятного исхода сборка завершится успешно. Нажав на Success можно увидеть информацию о сборке (Рис. 90).

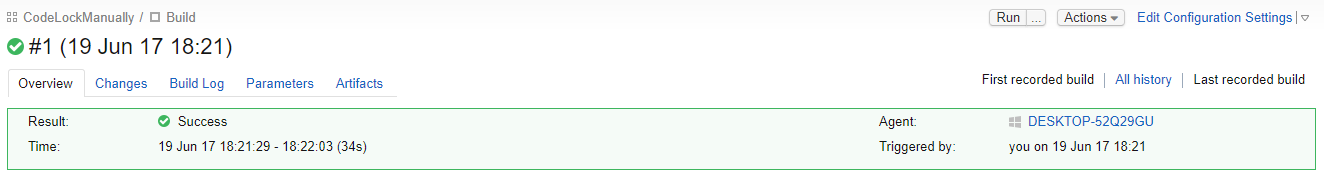


Рисунок 90 – Информация о сборке

В случае возникновения изменений в проекте можно нажать на кнопку Run и запустить новую сборку (Рис. 91).

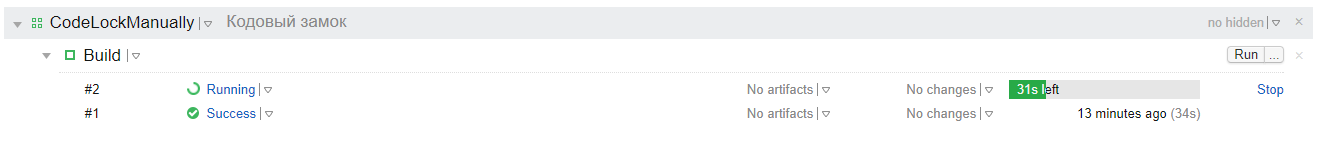


Рисунок 91 – Выполнение новой сборки

Также существует другой способ запуска сборки. После возникновения изменений, необходимо переместить изменённые файлы в область подготовленных файлов, совершить коммит и выгрузить данные в репозиторий. Таким образом, TeamCity, ранее связанный с GitHub, автоматически начнёт выполнять новую сборку (Рис. 92).

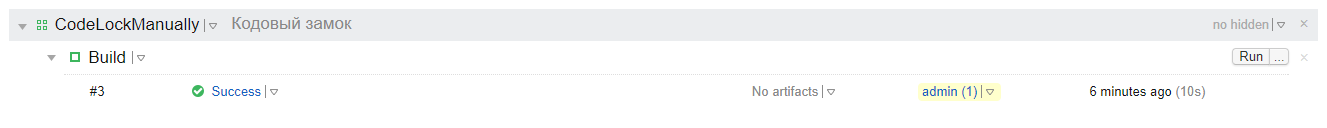


Рисунок 92 – Автоматическое выполнение сборки

TeamCity может отслеживать ошибки, которые может допустить пользователь. TeamCity уведомляет об этом пользователя и статус сборки меняется с Success на Error. Также TeamCity указывает какая ошибка была произведена (Рис. 93).

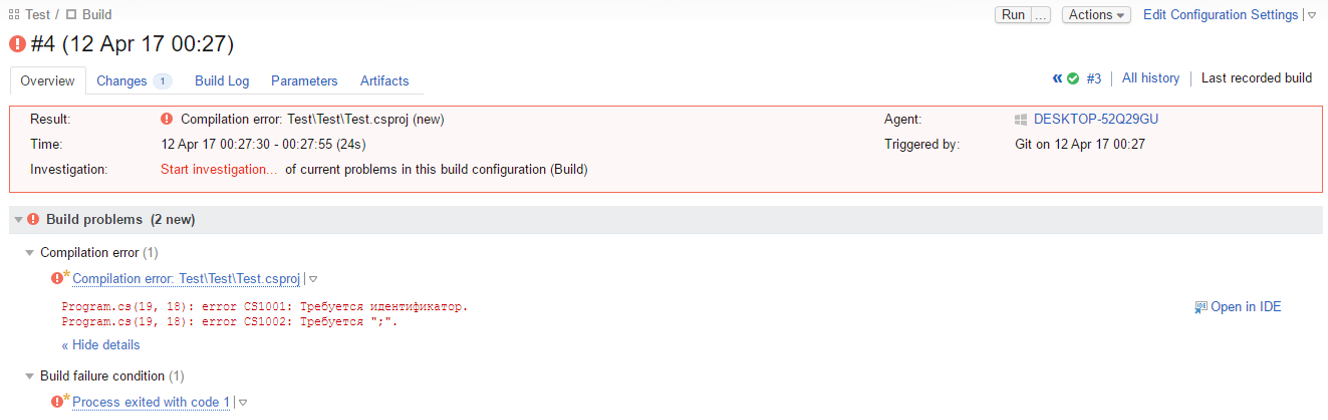


Рисунок 93 – Вывод ошибки

Кроме того, TeamCity может отслеживать ветки проекта (Рис. 94). Таким образом, можно получить информацию о том, в какой ветке была совершена ошибка.

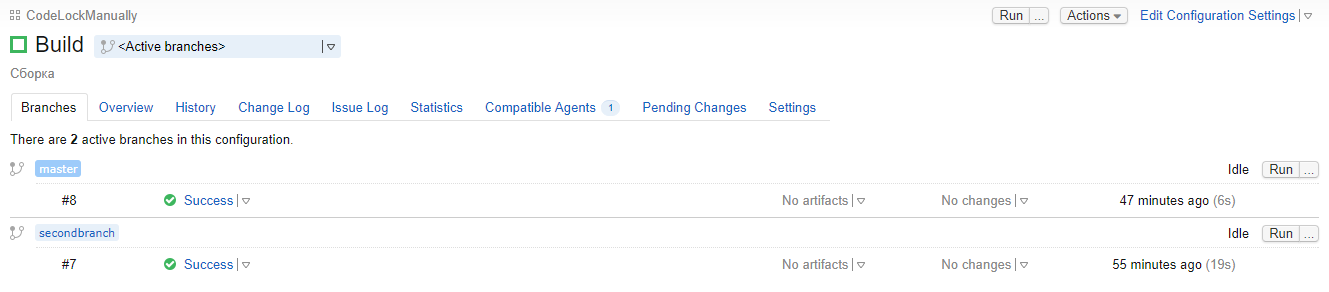


Рисунок 94 – Отслеживание ветвей проекта

В TeamCity присутствует формирование персональной сборки. Для её выполнения необходимо скачать плагин для используемой IDE и VCS. В случае ошибки персональная сборка не выгрузится на сервер. Таким образом, пользователь сможет исправить выявленную ошибку, без затрагивания исправной сборки.