

Лабораторная работа № 5. Работа с задачами через GitLab.

Цель работы

Получение навыков работы с системой отслеживания ошибок на примере GitLab Issue Tracking.

Внимание: для выполнения лабораторной вам будет необходимо установить IDE IntelliJ IDEA. Подробные инструкции приведены в документе «Настройка ПО для УРПО».

1. Теоретическая часть

1.1. Система отслеживания ошибок

Система отслеживания ошибок (англ. bug tracking system) — прикладная программа, разработанная с целью помочь разработчикам программного обеспечения (программистам, тестировщикам и прочим) учитывать и контролировать ошибки (баги), найденные в программах, пожелания пользователей, а также следить за процессом устранения этих ошибок и выполнением или невыполнением пожеланий.

В программировании *баг* (англ. bug — жук) — жаргонное слово, обычно обозначающее ошибку в программе или системе, которая выдаёт неожиданный или неправильный результат.

1.2. Состав информации о дефекте

Главный компонент системы отслеживания ошибок — база данных, содержащая сведения об обнаруженных дефектах. Эти сведения могут включать в себя:

- номер (идентификатор) дефекта;
- кто сообщил о дефекте;
- дата и время, когда был обнаружен дефект;
- версия продукта, в которой обнаружен дефект;
- серьёзность (критичность) дефекта и приоритет решения;
- описание шагов для выявления дефекта (воспроизведения неправильного поведения программы);
- кто ответственен за устранение дефекта;
- обсуждение возможных решений и их последствий;
- текущее состояние (статус) дефекта;
- версия продукта, в которой дефект исправлен.

Кроме того, развитые системы предоставляют возможность прикреплять файлы, помогающие описать проблему (например, дампы памяти или снимки экрана).

1.3. Жизненный цикл дефекта

Как правило, система отслеживания ошибок использует тот или иной вариант «жизненного цикла» ошибки, стадия которого определяется текущим состоянием, или статусом, в котором находится ошибка.

Типичный жизненный цикл дефекта:

1. Новый — дефект зарегистрирован тестировщиком.
2. Назначен — назначен ответственный за исправление дефекта.

3. Разрешён — дефект переходит обратно в сферу ответственности тестировщика.

2. Работа с GitLab

2.1. Fork проекта

Можно воспринимать форк проекта как копию чужого репозитория у себя. Чтобы сделать форк в GitLab, нужно просто перейти на страницу с проектом, и нажать «Fork».

2.2. Заведение дефекта.

Для заведения дефекта во вкладке проекта выберите «Issues», затем «New Issue». Внешний вид требуемого элемента показан на рисунке 6.

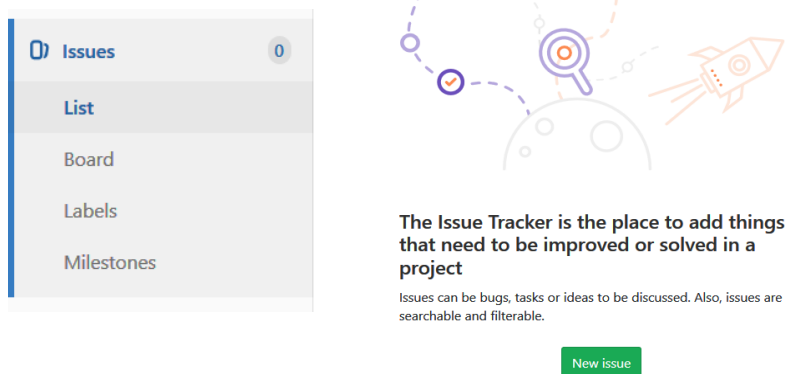


Рисунок 6 – Вкладка «Issues» проекта в GitLab

Создание новой задачи

Ниже рассмотрим пример заведения задачи/баги в трекере.

Во-первых, необходимо проставить название задачи, затем её описание.

Во-вторых, в лабораторной вам также нужно будет указать Assignee: человека, на которого назначена задача.

В-третьих, due date – время до следующей лабораторной. Остальные поля заполняются по желанию. Пример заведения задание в Issue Tracker'e сервиса GitLab показан на рисунке 7. Вид уже заведенной задачи продемонстрирован на рисунке 8.

New Issue

Title:

Add description templates to help your contributors communicate effectively!

Description:

Write Preview

В методе **** ошибка, поправьте! Ничего не работает, сроки горят!

Markdown and quick actions are supported

[Attach a file](#)

☐ This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee:

Due date:

Milestone:

Labels:

[Submit issue](#) [Cancel](#)

Рисунок 7 – Пример заведения ошибки в Issue Tracker'e GitLab

[Open](#) Opened 21 seconds ago by [Топраева Ксения Олеговна](#)

[Close issue](#) [New issue](#)

Новая бага

В методе **** ошибка, поправьте! Ничего не работает, сроки горят!

[👍 0](#) [👎 0](#) [😊](#) [Create merge request](#)

[Write](#) Preview

Write a comment or drag your files here...

Markdown and quick actions are supported

[Attach a file](#)

[Comment](#) [Close issue](#)

Рисунок 8 – Вид заведённой задачи

2.3. Создание ветви для дефекта

Далее вам будет нужно создать ветку для решения этой задачи. Это можно сделать на странице с задачей, в окне «Create branch» (рисунок 9).

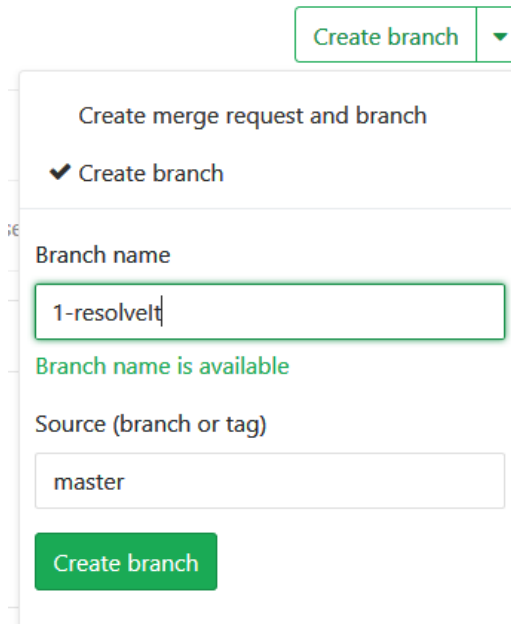


Рисунок 9 – Создание ветки, соотносящейся с задачей

2.4. Клонирование репозитория

Операции клонирования репозитория, перехода на ветку, коммита и отправки на удалённый сервер рассматривались ранее, поэтому здесь мы их рассмотрим кратко.

Для клонирования используется команда `git clone <URL>` (рисунок 10). URL можно найти на странице с проектом (Fork), который вы сделаете.

```
$ git clone https://sgit.rsreu.ru/g640_torgaeva.x.o/test-project.git
Cloning into 'test-project'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 2), reused 0 (delta 0)
Unpacking objects: 100% (9/9), done.
```

Рисунок 10 – Клонирование проекта

1.4.5. Прочие необходимые операции

Далее вам нужно будет перейти на нужную ветку командой `git checkout <branch>`, где `branch` – ветка, которую вы создали. Затем исправить ошибку, описанную вашим напарником.

Далее нужно отправить изменения на удалённый сервер. Для этого добавьте изменённый файл к будущему коммиту (`git add <filename>`), и

сделайте коммит (`git commit -m «Осмысленное сообщение коммита»`) (рисунок 11). Далее отправьте изменение на удалённый сервер с помощью команды `git push` (рисунок 12).

```
$ git commit -m "Исправление баги"
[1-resolveIt 80867b2] Исправление баги
1 file changed, 1 insertion(+)
create mode 100644 file.txt
```

Рисунок 11 – Фиксирование изменений

```
$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 322 bytes | 107.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: To create a merge request for 1-resolveIt, visit:
remote:   https://sgit.rsreu.ru/g640_torgaeva.x.o/test-project/merge_requests/new?merge_request%5Bsource_branch%5D=1-resolveIt
remote:
To https://sgit.rsreu.ru/g640_torgaeva.x.o/test-project.git
   1de851b..80867b2  1-resolveIt -> 1-resolveIt
```

Рисунок 12 – Отправка изменений на удалённый сервер

Практическая часть

Замечание: замените * на номер вашего варианта.

Ход работы:

1. Сделайте форк из репозитория <https://lexie62rus@bitbucket.org/lexie62rus/otslezhivanie-oshibok.git>.

2. Перейдите в нужный пакет (`util/variant_*`). Модули `Parser_1` и `Parser_2` для вас и напарника соответственно. Вы и ваш напарник, найдите ошибку в выбранном методе (одна ошибка в методе, см. таблицу 1 ниже). Ошибки можно вычислить по логике, или по несовпадению кода с комментарием к нему.

3. Заведите задачу в GitLab, и назначьте ее исполнение на напарника.

4. Создать ветку с любым именем на странице с заведённой задачей `issue tracker'a`.

5. Клонировать проект на локальный компьютер.

6. Исправьте ошибку в этой созданной ветке, проверьте и добавьте скриншоты и примеры.

7. Сделайте коммит. Залейте свои изменения на удалённый сервер.

8. Закройте задачу через GitLab.

Таблица 1 – Варианты заданий (названия методов с ошибками)

Вариант	Метод
	Calc (1.1 и 1.2)
	fixNegativeValue (2.1. и 2.2)
	deleteSpaces (3.1) isOperationSign (3.2)
	fixMultipleSignLack (4.1. и 4.2)
	openBracketsAndCalculateExpression (5.1. и 5.2)

Содержание отчёта

По результатам выполнения работы оформляется отчет в соответствии с требованиями ГОСТ 7.32-2017 «Отчет о научно-исследовательской работе. Структура и правила оформления», включающий:

- титульный лист;
- цель работы;
- описание хода выполнения работы с подробными комментариями и полученные результаты;
- выводы.

Контрольные вопросы:

1. Что такое система отслеживания ошибок?
2. Что такое баг?
3. Что такое crash report?
4. Опишите основные составляющие информации о дефекте.
5. Опишите жизненный цикл дефекта.
6. Назовите несколько известных вам систем отслеживания ошибок.
7. Опишите последовательность работы с дефектом с использованием GitLab.