

ЛАБОРАТОРНАЯ РАБОТА № 4. "Git. Серверная часть."

1. Цель работы

Получение навыков работы с системой контроля версий Git, используя GitLab (ветки, разрешение конфликтов).

Продолжительность работы: 2 часа.

Работа выполняется в парах.

2. Теоретическая часть

2.1. Ветвление

Ветка - это направления разработки, которое существует независимо от другого направления, однако имеющие с ним общую историю, если заглянуть немного в прошлое. Ветка всегда берет начало как копия чего-либо и движется от этого момента создавая свою собственную историю.

Ветвление Git очень легковесно. Операция создания ветки выполняется почти мгновенно, переключение между ветками - также быстро. В отличие от многих других СКВ, Git поощряет процесс работы, при котором ветвление и слияние выполняется часто, даже по несколько раз в день.

Ветка (branch) в Git — это легко перемещаемый указатель на один из коммитов. Имя основной ветки по умолчанию в Git — master.

Когда вы делаете коммиты, то получаете основную ветку, указывающую на ваш последний коммит. Каждый коммит автоматически двигает этот указатель вперед.

Что же на самом деле происходит, когда вы создаете ветку? Всего лишь создается новый указатель для дальнейшего перемещения. Допустим вы хотите создать новую ветку с именем “testing” Вы можете это сделать командой **git branch** :

Пример 1. Создание ветки testing. При создании мы останемся на той ветке, на которой были:

```
$ git branch testing
```

Чтобы переключиться на существующую ветку, выполните команду **git checkout**. Давайте переключимся на ветку “testing”:

Пример 2. Перемещение на ветку testing:

```
$ git checkout testing
```

Слияние веток

Слияние веток выполняется с помощью команды **git merge <branchname>**. Например, если мы хотим влить изменения из ветки iss53 в текущую ветку, в которой мы находимся, необходимо выполнить:

Пример 3. Вливание изменений из ветки iss53 в текущую ветку (на которой мы сейчас находимся):

```
$ git merge iss53
Updating f483254..3a0874c
Fast forward
 README /      1-
 1 file changed, 0 insertions( + ), 1 deletions( - )
```

Узнать ветку, на которой мы находимся, можно командой `git status`.

2.2. Просмотр истории изменений с ветвлениями в удобном виде

Вы можете увидеть историю ревизий помощи команды `git log`. Команда `git log --oneline --decorate --graph --all` выдаст историю ваших коммитов и покажет, где находятся указатели ваших веток, и как ветвилась история проекта.

Пример 4. Просмотр изменений с ветвлениями в удобном виде:

```
$ git log --oneline --decorate --graph --all
* c2b9e (HEAD, master) made other changes
/ * 87ab2 (testing) made a change
//
*f30ab add feature #32 - ability to add new formats to the
* 34ac2 fixed bug #1328 - stack overflow under certain conditions
* 98ca9 initial commit of my project
```

2.3. Разрешение конфликтов слияния

Иногда процесс не проходит гладко. Если вы изменили одну и ту же часть одного и того же файла по-разному в двух объединяемых ветках, Git не сможет самостоятельно объединить их. Если ваше исправление ошибки #53 потребовало изменить ту же часть файла, что и hotfix, вы получите примерно такое сообщение о конфликте слияния:

Пример 5. Конфликт при слиянии изменений:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

Git не создал коммит слияния автоматически. Он остановил процесс до тех пор, пока вы не разрешите конфликт. Чтобы в любой момент после появления конфликта увидеть, какие файлы не объединены, вы можете запустить `git status`:

```
$ git status
On branch master
```

*You have unmerged paths.
(fix conflicts and run "git commit")*

*Unmerged paths:
(use "git add <file>..." to mark resolution)
both modified: index.html
no changes added to commit (use "git add" and/or "git commit -a")*

Всё, где есть неразрешенные конфликты слияния, перечисляется как неслитое. Git добавляет в конфликтующие файлы стандартные пометки разрешения конфликтов, чтобы вы могли вручную открыть их и разрешить конфликты.

В вашем файле появился раздел, выглядящий примерно так:

Пример 6. Файл с маркерами конфликта:

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```

Способы разрешения конфликта в git такие же, как и в Subversion:

- Применить чужое изменения:
git merge -X theirs branch;
- Применить свои изменения:
git merge -s ours;
- Вручную объединить файлы (посредством ручного редактирования файла с метками конфликта).

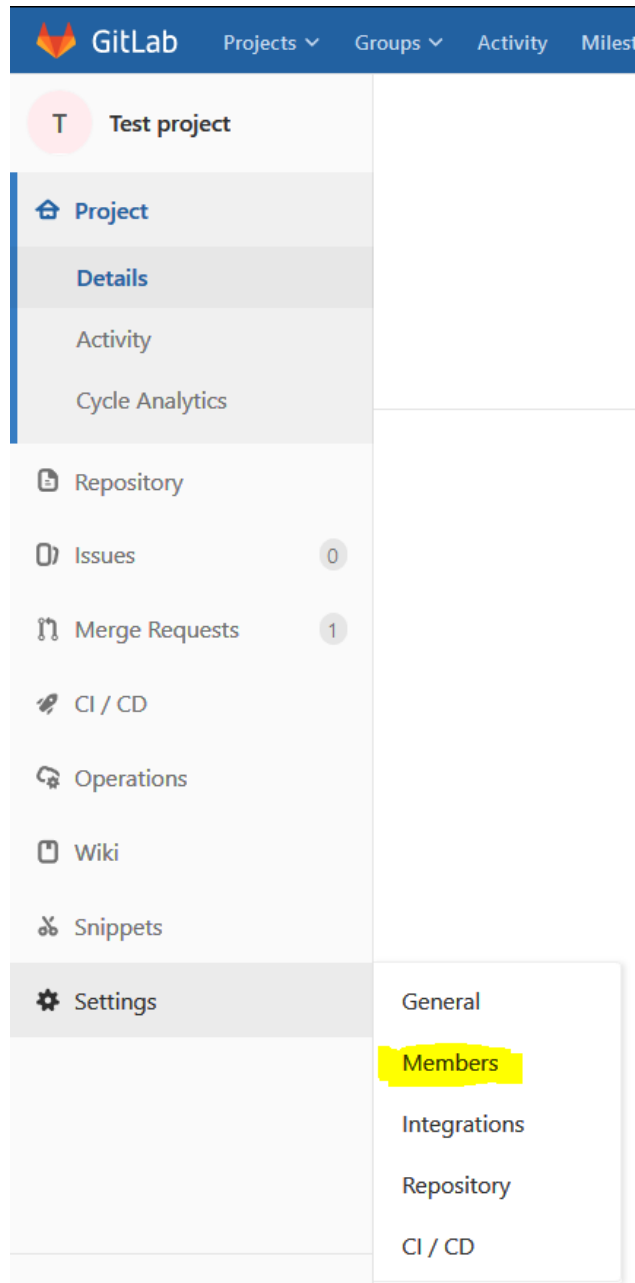
Как только вы завершили слияние, зафиксируйте изменения в репозитории командой `git commit`.

2.4. Серверная работа с Git

2.4.1. Совместная работа

Самый простой метод совместной работы над проектом GitLab — это выдача другому пользователю прямого доступа на запись (push) в git-репозитории. Вы можете добавить пользователя в проект в разделе “Участники” (“Members”) настроек проекта, указав уровень доступа. Получая уровень доступа “Разработчик” (“Developer”) или выше, пользователь может отсылать свои коммиты и ветки непосредственно в репозиторий.

Для добавления зайдите в настройки проекта Settings-Members. Выберите способ участия Developer. Дату окончания участия оставьте на своё усмотрение.



Настройки проекта

Project members

You can invite a new member to **Test project** or invite another group.

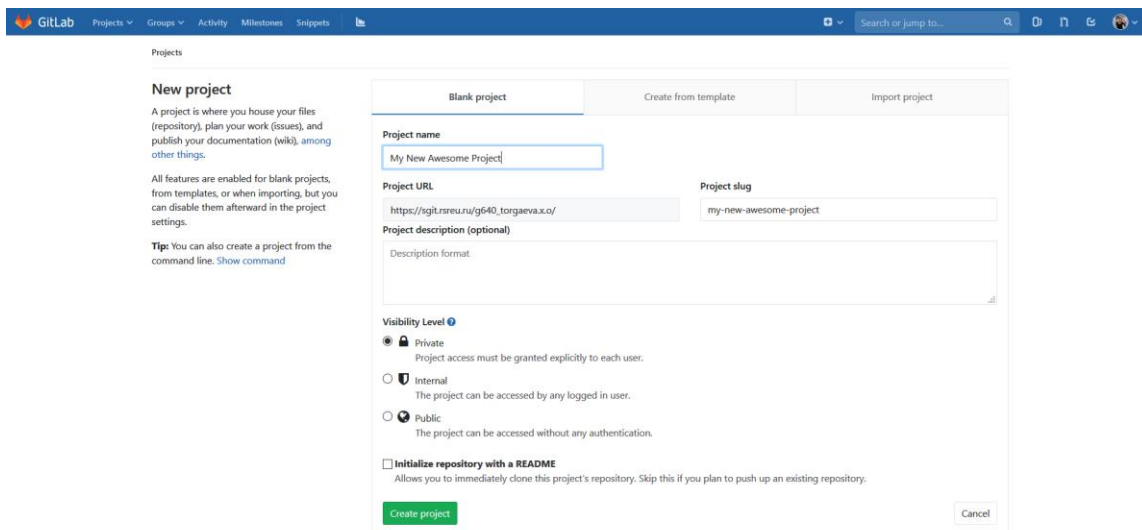
Invite member	Invite group
Select members to invite	
<input type="text" value="Administrator"/>	
Choose a role permission	
<input type="text" value="Developer"/>	
Read more about role permissions	
Access expiration date	
<input type="text" value="2018-12-26"/>	
<input type="button" value="Add to project"/>	<input type="button" value="Import"/>

Окно добавления участника в проект

2.4.2. Создание репозитория

Рассмотрим пример создания нового проекта:

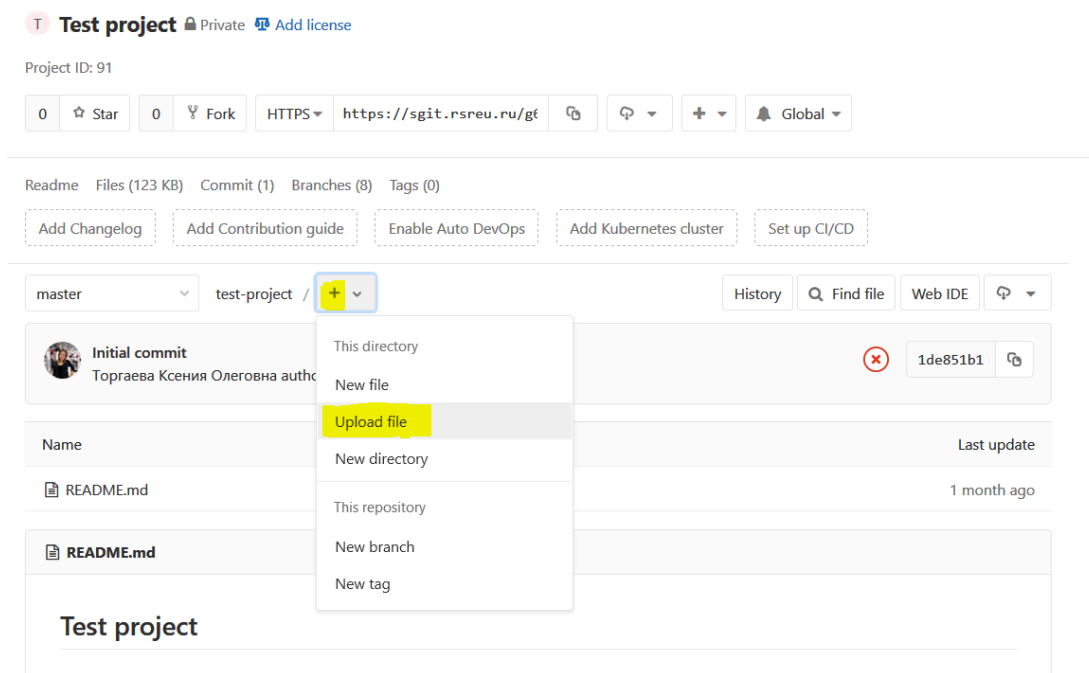
В окне создания проекта можно настроить его имя, URL, описание проекта, является ли он публичным и приватным. Также вам будет предложено проинициализировать проект, добавив в него файл README.

The screenshot shows the 'New project' form in GitLab. It has three tabs: 'Blank project', 'Create from template', and 'Import project'. The 'Blank project' tab is active. The form includes fields for 'Project name' (filled with 'My New Awesome Project'), 'Project URL' (filled with 'https://git.rseu.ru/g640_torgaeva.x.o/'), and 'Project slug' (filled with 'my-new-awesome-project'). There is a 'Project description (optional)' text area. Below these is the 'Visibility Level' section with three radio buttons: 'Private' (selected), 'Internal', and 'Public'. A note under 'Private' says 'Project access must be granted explicitly to each user.' Below that is a checkbox 'Initialize repository with a README' with a note 'Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.' At the bottom are 'Create project' and 'Cancel' buttons.

Окно создания нового проекта

1.4.3. Добавление файлов в репозиторий через GitLab

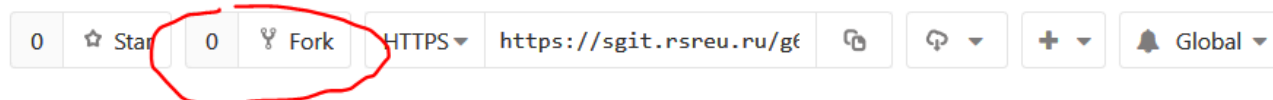
Для добавления файла зайдите в созданный проект (Project -> Details) и нажмите «+», Upload file (загрузить файл), и выберите нужный файл.

The screenshot shows the 'Test project' details page in GitLab. At the top, it says 'Test project' with 'Private' and 'Add license' links. Below is 'Project ID: 91' and a row of buttons: '0 Star', '0 Fork', 'HTTPS', 'https://git.rseu.ru/g640_torgaeva.x.o/', a clone icon, a dropdown menu with a '+' icon, and a 'Global' button. Below this is a row of tabs: 'Readme', 'Files (123 KB)', 'Commit (1)', 'Branches (8)', and 'Tags (0)'. Under the tabs are several buttons: 'Add Changelog', 'Add Contribution guide', 'Enable Auto DevOps', 'Add Kubernetes cluster', and 'Set up CI/CD'. Below the buttons is a section for the 'master' branch. It shows 'test-project /' with a dropdown menu open. The dropdown menu has options: 'This directory', 'New file', 'Upload file' (highlighted in yellow), 'New directory', 'This repository', 'New branch', and 'New tag'. To the right of the dropdown is a 'History' button, a 'Find file' search bar, and a 'Web IDE' button. Below the dropdown is a table with columns 'Name' and 'Last update'. The first row shows 'Initial commit' by 'Торгаева Ксения Олеговна' with a commit hash '1de851b1' and a date '1 month ago'. Below the table is a 'Test project' section.

Добавление файла через GitLab

2.4.3. 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».



Fork проекта

2.4.4. Клонирование проекта (создание рабочей копии)

Сразу после этого можно будет создать рабочую копию на своём компьютере и работать с проектом – для этого перейдите на страницу с проектом и нажмите “clone”, скопируйте url и создайте рабочую копию репозитория в выбранной вами папке с помощью команды **git clone <url>**.

2.5. Изменение последнего сообщения коммита

Бывают ситуации, когда вы неверно написали сообщение к коммиту. Вы можете установить сообщение к последнему коммиту непосредственно в командной строке:

Пример 7. Редактирование сообщение последнего коммита:

```
git commit --amend -m "New commit message"
```

Перед тем, как сделать это, убедитесь, что у вас нет каких-либо изменений в рабочей копии, иначе они также будут совершены. (Нестационарные изменения не будут зафиксированы.)

2.6. «Забирание» отдельного коммита

Команда `git cherry-pick` используется для того чтобы взять изменения, внесённые каким-либо коммитом, и попытаться применить их заново в виде нового коммита наверху текущей ветки. Это может оказаться полезным чтобы забрать парочку коммитов из другой ветки без полного слияния с той веткой.

Пример 8. «Забирание» коммита с хеш-кодом «e43a6fd3e...» в текущую ветку:

```
$ git cherry-pick e43a6fd3e94888d76779ad79fb568ed180e5fcd
Finished one cherry-pick.
[master]: created a0a41a9: "More friendly message when locking the index fails."
3 files changed, 17 insertions(+), 3 deletions(-)
```

3. Практическая часть

Практическая часть:

Замечание: замените * на номер вашего варианта. Забирайте необходимые изменения у напарника и отправляйте их на удалённый сервер, когда необходимо.

1. Один из вас создаёт репозиторий, добавляет туда файла fixme.cpp (находится в этой же директории ЛР_4).
2. Участник, создавший репозиторий, добавляет второго в коллабораторы проекта через GitLab.
3. Первый и второй участник клонируют проект на компьютеры.
4. Создайте ветки по шаблону feature/variant_*_[имя].
5. Поправьте файл "fixme.cpp". Сделайте так, чтобы ваши изменения покрывали одни и те же участки кода.
6. Влейте изменения ветки напарника к себе в ветку. Отрадите в сообщении к коммиту, то, что это коммит с вливанием веток.
7. Разрешите возникшие конфликты.
8. Исследуйте, когда конфликт получается, а когда - нет.
9. Влейте изменения в ветку feature/develop_* (предварительно её создав).
10. Сделайте ещё два любых коммита в своих ветках.

Задания для продвинутого уровня:

- редактирование последнего сообщения коммитов;
- оба напарника делают коммит, каждый забирает его отдельно, без вливания ветки;
- влейте в свою ветку ветку напарника и затем отмените коммит со слиянием веток.

4. Содержание отчёта

- титульный лист;
- цель работы;
- добавление участника в союзники с помощью GitLab;
- описание структуры хранилища во время выполнения (при выполнении операций, меняющих состояние хранилища. Для документирования состояния хранилища можно использовать команду для удобного просмотра лога из п.2.2.);
- выполняемые команды с комментариями и результаты их выполнения;
- вывод.

5. Контрольные вопросы:

1. Отличие локального репозитория от удалённого.
2. Способы копирования проекта к себе в удалённый репозиторий.
3. Как отредактировать сообщение у коммита?
4. Что такое ветка?
5. Что представляет из себя ветка в Subversion?
6. Недостатки работы в одной ветке.
7. Как скопировать отдельные изменения между ветками?
8. Как скопировать изменения из одной ветки в другую?
9. Какие файлы создаются при конфликтной ситуации при слиянии?
10. Способы разрешения конфликтов в Subversion.