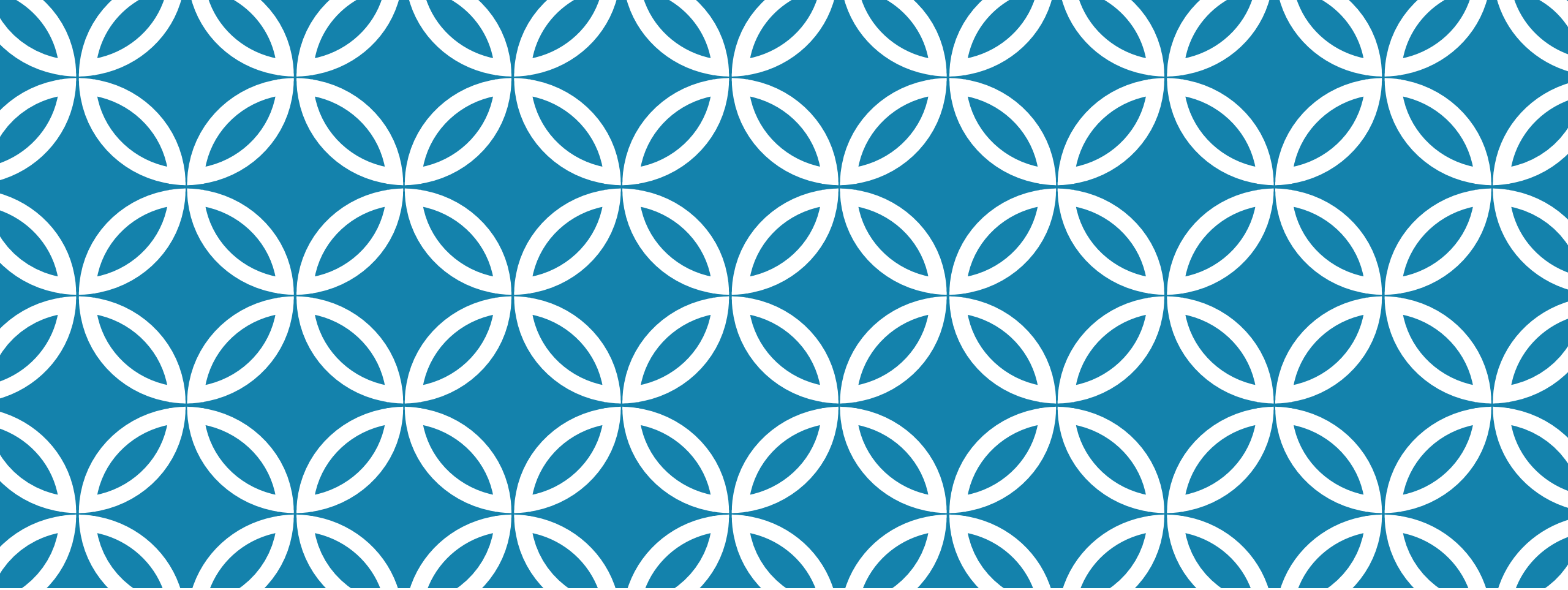


ЛЕКЦИЯ 4

Git и Subversion.
Работа на сервере

ЧТО БУДЕМ РАССМАТРИВАТЬ

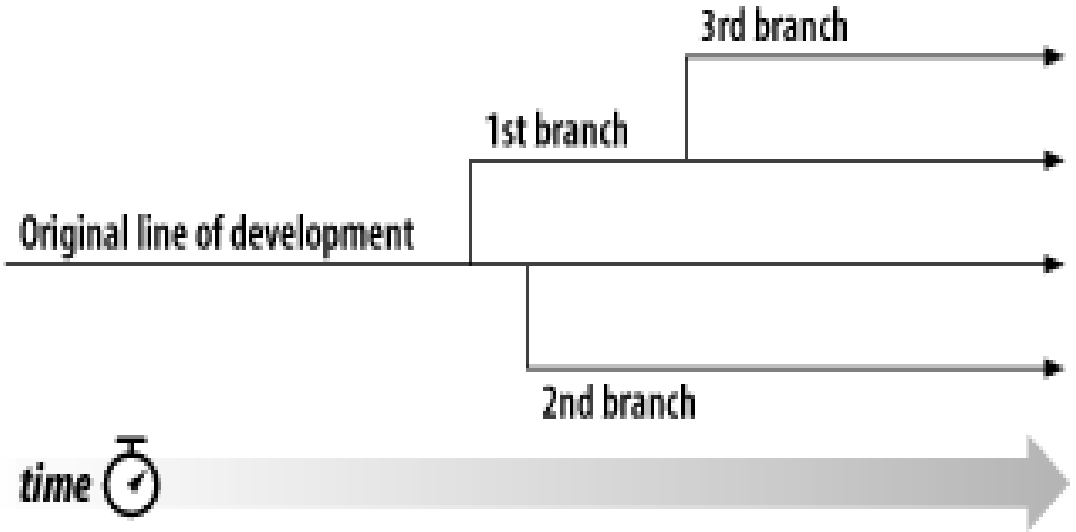
- Ветвление в Subversion и Git
- Слияние изменений в Subversion и Git
- Работа с GitLab

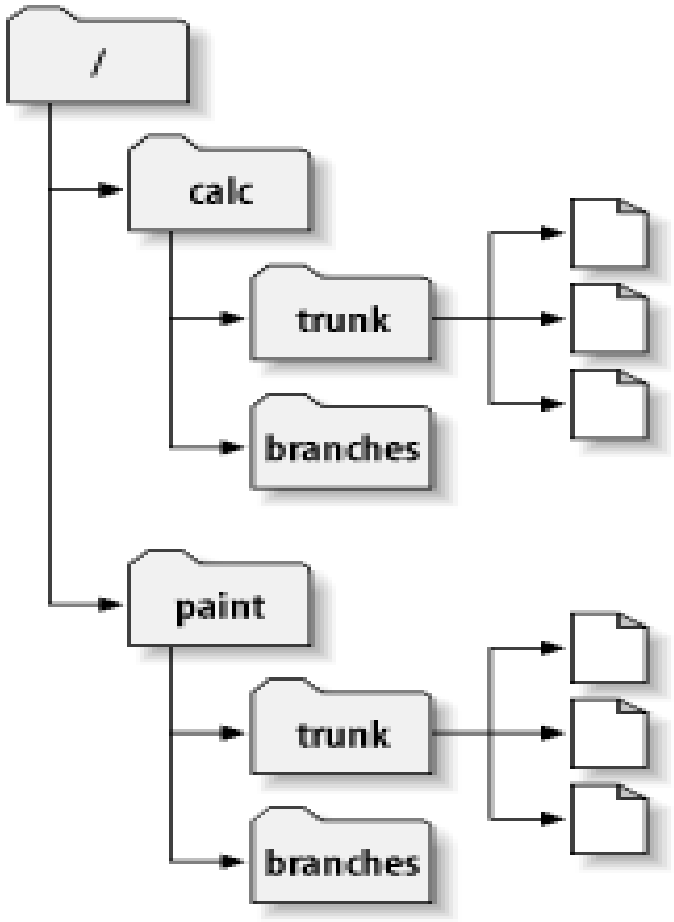
Дополнительные темы:

- Open Source разработка с помощью GitHub
- Модель ветвления git flow
- Поиск ломающего коммита методом половинного деления

ВЕТВЛЕНИЕ В SUBVERSION

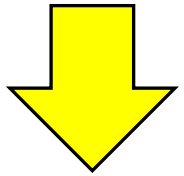
Ветка - это направления разработки, которое существует независимо от другого направления, однако имеющие с ним общую историю, если заглянуть немного в прошлое. Ветка всегда берет начало как копия чего-либо и двигается от этого момента создавая свою собственную историю.





НЕДОСТАТКИ РАБОТЫ В ОДНОЙ ВЕТВИ

- Это не надежно. Большинство людей предпочитают часто сохранять свою работу в хранилище, на случай если вдруг что-то плохое случится с рабочей копией;
- Это не достаточно гибко. Если вы работаете на разных компьютерах, вам придется вручную копировать изменения взад и вперед, либо делать всю работу на одном компьютере.



Необходимо использовать ветви

СОЗДАНИЕ ВЕТКИ

Создание ветки в Subversion выполняется довольно просто - при помощи команды svn copy делаете в хранилище копию проекта.

Есть два способа создания копии:

- С клонированием рабочей копии и созданием ветви;
- Просто с созданием новой ветви, которая вам необходима.

СОЗДАНИЕ ВЕТВИ С ПОМОЩЬЮ КЛОНИРОВАНИЯ РАБОЧЕЙ КОПИИ

Для начала, создается рабочая копия корневой директории проекта /calc:

$ svn checkout http://svn.example.com/repos/calc bigwc

$ svn copy trunk branches/my-calc-branch (вызывается из папки проекта)

$ svn status

A + branches/my-calc-branch

В этом случае, команда svn copy рекурсивно копирует рабочую директорию trunk в новую рабочую директорию branches/my-calc-branch.

СОЗДАНИЕ ВЕТВИ С ПОМОЩЬЮ URL

Svn copy может оперировать с двумя URL напрямую

*$ svn copy http://svn.example.com/repos/calc/trunk *

*http://svn.example.com/repos/calc/branches/my-calc-branch *

-m "Creating a private branch of /calc/trunk."

Committed revision 341.

РАЗЛИЧИЯ МЕЖДУ ДВУМЯ МЕТОДАМИ

В сущности, между этими двумя методами нет разницы.

Но во втором:

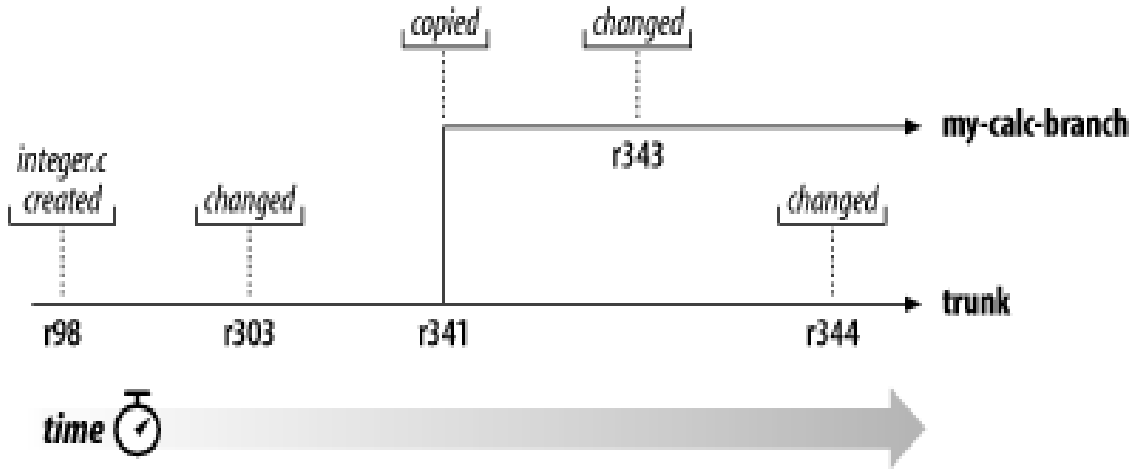
- Выполняется немедленная фиксация
- Эта процедура более проста в использовании, так как нет необходимости в создании рабочей копии, отражающей большое хранилище. В этом случае вам вовсе можно не иметь рабочей копии.

РАБОТА С ВЕТКОЙ

После создания ветки проекта, можно создать новую рабочую копию для начала ее использования:

$ svn checkout http://svn.example.com/repos/calc/branches/my-calc-branch

В этой рабочей копии нет ничего особенного. Она является просто отражением другой директории хранилища.



КЛЮЧЕВЫЕ ИДЕИ, СТОЯЩИЕ ЗА ВЕТКАМИ

- В отличие от других систем управления версиями, ветки в Subversion существуют в хранилище не в отдельном измерении, а как обычные нормальные директории файловой системы. Такие директории просто содержат дополнительную информацию о своей истории;
- Subversion не имеет такого понятия как ветка — есть только копии. При копировании директории результирующая директория становиться «веткой» только потому что вы рассматриваете ее таким образом.

КОПИРОВАНИЕ ИЗМЕНЕНИЙ МЕЖДУ ВЕТКАМИ

В проектах, имеющих большое количество участников, стандартной процедурой является создать отдельную ветку и фиксировать изменения туда пока работа не будет полностью завершена.

Чтобы избежать слишком большого расхождения веток, можно продолжать делиться изменениями по ходу работы. Когда ваша ветка будет полностью закончена, полный набор изменений ветки может быть скопирован обратно в основную ветку.

КОПИРОВАНИЕ ОТДЕЛЬНЫХ ИЗМЕНЕНИЙ

svn merge способна сравнивать любые два объекта в хранилище и показывать изменения, а также их применять.

Например, вы можете попросить svn merge применить правку 344 к рабочей копии в виде локальных изменений:

$ svn merge -r 343:344 http://svn.example.com/repos/calc/trunk

U integer.c

$ svn status

M integer.c

(при слиянии файл может оказаться в состоянии конфликта. В этом случае нужно решить его одним из рассмотренных на предыдущем занятии способом)

SVN MERGE

После просмотра результата объединения изменений, можно их как обычно зафиксировать (svn commit).

После этого изменения будут внесены в вашу ветку хранилища. В терминах контроля версий такую процедуру копирования изменений между ветками обычно называют *портированием изменений (слиянием, “мерджем” (merge))*.

КЛЮЧЕВЫЕ ПОНЯТИЯ, СТОЯЩИЕ ЗА СЛИЯНИЕМ (SVN MERGE)

Понять как именно ведет себя svn merge достаточно просто.

В замешательство приводит, главным образом название команды. Термин «слияние» как бы указывает на то, что ветки соединяются вместе, или происходит какое-то волшебное смешивание данных. На самом деле это не так. Лучшим названием для этой команды могло быть svn diff-and-apply потому что это все, что происходит: сравниваются два файловых дерева хранилища, а различия переносятся в рабочую копию.

КЛЮЧЕВЫЕ ПОНЯТИЯ, СТОЯЩИЕ ЗА СЛИЯНИЕМ (SVN MERGE)

Команда svn merge принимает три аргумента:

1. Начальное дерево хранилища (как правило, называемое левой частью при сравнении),

2. Конечное дерево хранилища (как правило называемое правой частью при сравнении),

3. Рабочую копию для применения отличий, в виде локальных изменений (как правило, называемую целью слияния).

Когда эти три аргумента указаны, сравниваются два дерева и результирующие различия применяются к целевой рабочей копии в виде локальных изменений. Если результат вас не устраивает, просто отмените (svn revert) все сделанные изменения.

Лучше всего её было бы назвать diff-and-apply, т.к. она делает именно это.

КОНФЛИКТЫ ПРИ ОБЪЕДИНЕНИИ (ОТЛИЧИЯ ОТ SVN UPDATE)

- svn merge иногда не может гарантировать правильного и может вести себя хаотично: пользователь может запросить сервер сравнить любые два дерева файлов, даже такие, которые не имеют отношения к рабочей копии! Из этого следует большое количество потенциальных человеческих ошибок.
- названия файлов, создаваемых при возникновении конфликта: filename.working, filename.left и filename.right (вместо filename.rOLDREV и filename.rNEWREW при svn update).

УЧИТЫВАТЬ ИЛИ ИГНОРИРОВАТЬ ПРОИСХОЖДЕНИЕ

При общении разработчиков, использующих Subversion очень часто можно услышать упоминание термина *происхождение*. Это слово используется для описания отношений между двумя объектами хранилища: если между ними есть связь, тогда говорят, что один объект является предком другого.

svn diff игнорирует происхождение, в то время, как svn merge его учитывает. Данный аспект поведения обеих команд можно переопределять. Подробнее об этом в svn help.

ВЕТВЛЕНИЕ В GIT

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.

СОЗДАНИЕ ВЕТКИ

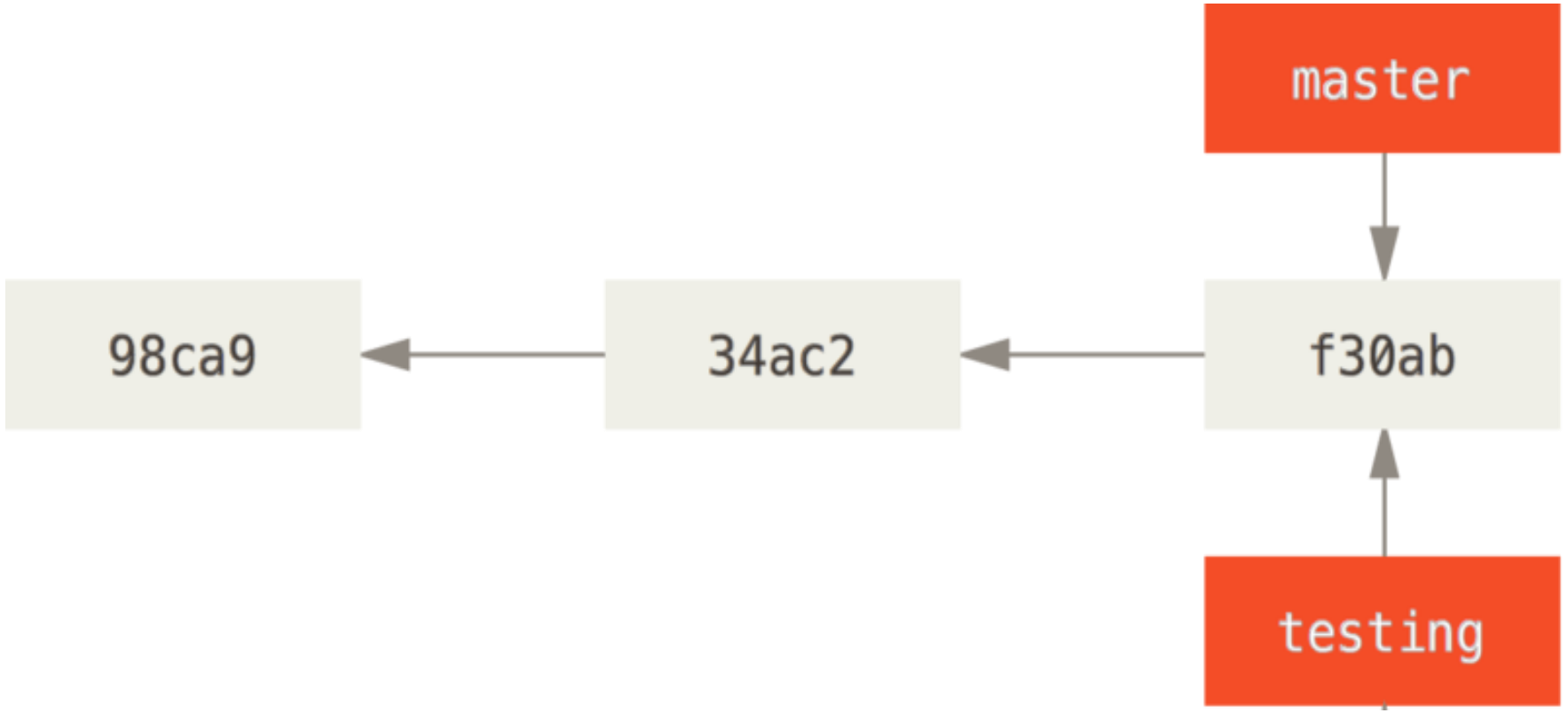
Допустим вы хотите создать новую ветку с именем “testing” Вы можете это сделать командой git branch <branchname>:

$ git branch testing

По умолчанию команда *git branch* создаёт новую ветку, но не переключается на неё.

В результате создается новый указатель на тот же самый коммит, в котором вы находитесь.

СОЗДАННАЯ ВЕТКА TESTING



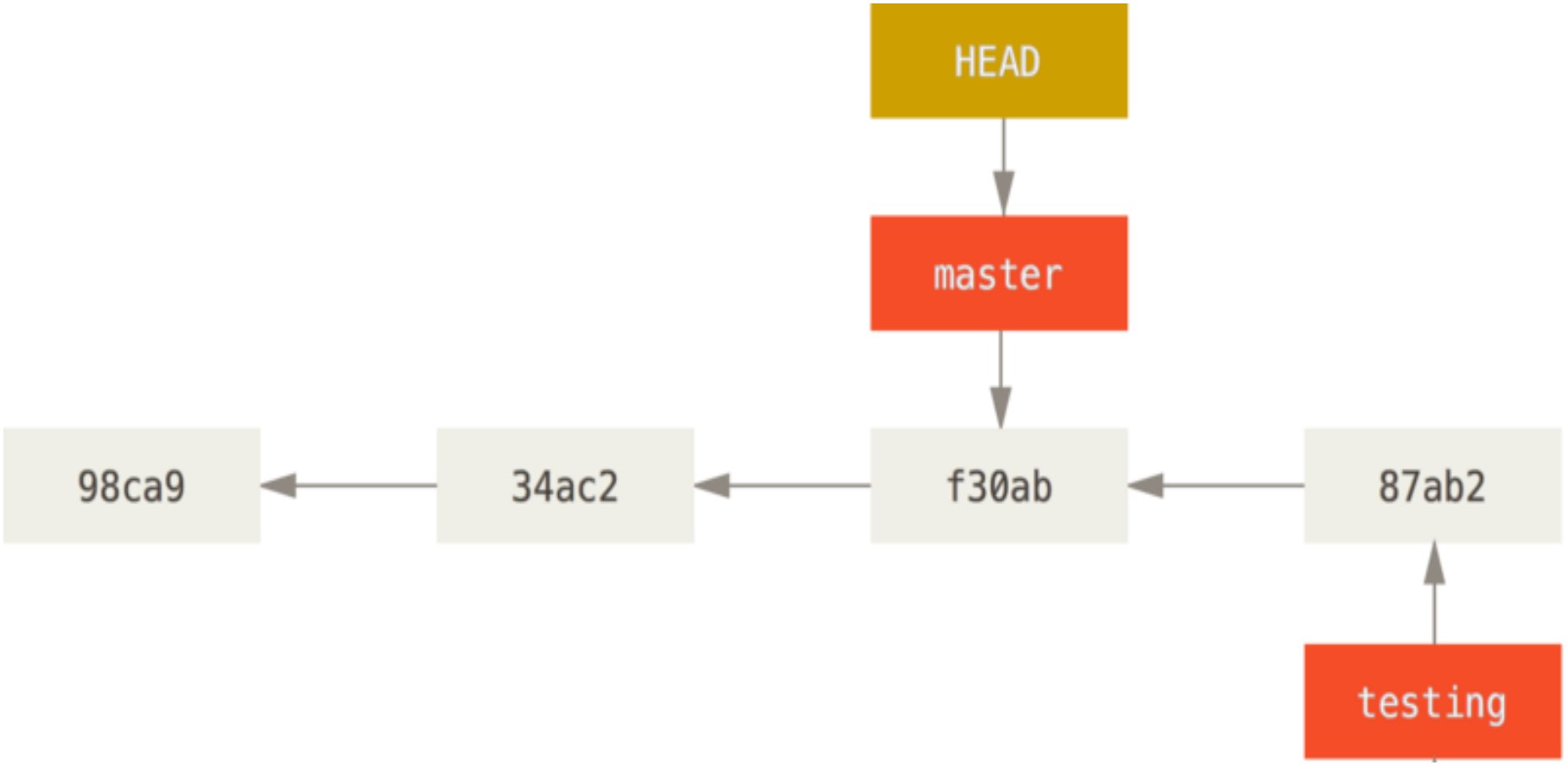
ПЕРЕКЛЮЧЕНИЕ НА ВЕТКУ TESTING

Чтобы переключиться на существующую ветку, выполните команду **git checkout <branchname>**:

$ git checkout testing

ПЕРЕКЛЮЧЕНИЕ НА ВЕТКУ MASTER

Если сделать ещё один коммит в ветке testing и перейдём на ветку master



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:

$ git merge iss53

Updating f483254..3a0874c

Fast forward

README / 1-

1 file changed, 0 insertions(+), 1 deletions(-)

ПРОСМОТР ИСТОРИИ ИЗМЕНЕНИЙ 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.

```
$ 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
```

ОСОБЕННОСТИ ВЕТВЛЕНИЯ В GIT

Создание и удаление веток совершенно быстро и не затратно, так как ветка в Git — это всего лишь файл, содержащий 40 символов контрольной суммы SHA-1 того коммита, на который он указывает.

Это совершенно отличает Git от ветвления в большинстве более старых систем контроля версий, где все файлы проекта копируются в другой подкаталог. Там ветвление для проектов разного размера может занять от секунд до минут. В Git ветвление всегда мгновенное. Эти возможности побуждают разработчиков чаще создавать ветки.

РАЗРЕШЕНИЕ КОНФЛИКТОВ СЛИЯНИЯ

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

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.

РАЗРЕШЕНИЕ КОНФЛИКТОВ СЛИЯНИЯ

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.

ВИД НЕСЛИТОГО ФАЙЛА

В вашем файле появился раздел, выглядящий примерно так:

<<<<<<< HEAD:index.html

* contact : email.support@github.com *

=======

* *

please contact us at support@github.com

* *

>>>>>>> iss53:index.html

СПОСОБЫ РАЗРЕШЕНИЯ КОНФЛИКТОВ

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 коммандой git commit.

СЕРВЕРНАЯ РАБОТА С GIT

Для организации совместной работы необходимо разместить (to host) файл на удалённом сервере. Внутри одной организации никто не мешает сделать этого на одном из компьютеров, но это не совсем удобно, т.к. работа будет зависеть от питания этого компьютера.

Для хостинга можно использовать свой сервер или использовать готовое решение. Самыми популярными в настоящее время являются GitHub, Bitbacket, GitLab и другие.

Мы в лабораторных работах будем работать с сервисом GitLab, развёрнутым в РГРТУ, и с ним познакомимся подробнее.

КАКИЕ ПРОТОКОЛЫ МОЖНО НАСТРОИТЬ НА СВОЁМ GIT-СЕРВЕРЕ?

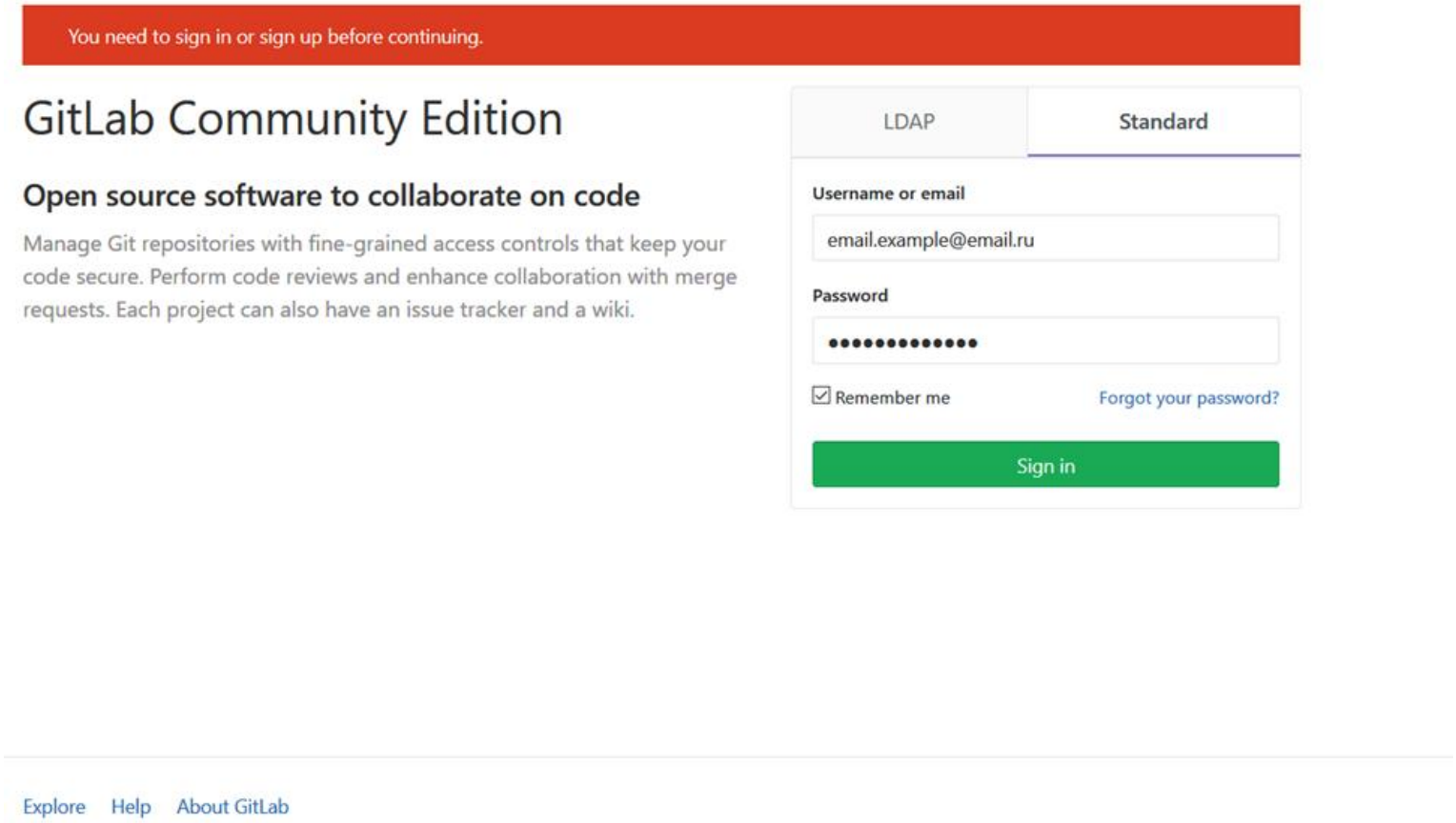
- HTTP – распространённый протокол, но сложнее остальных в настройке;
- SSH – легко настраивается, или присутствует на большинстве серверов, но отсутствует возможность анонимного доступа к репозиорию;
- Git – самый быстрый протокол, но не имеет аутенификации.

КАК РАБОТАТЬ СОВМЕСТНО?

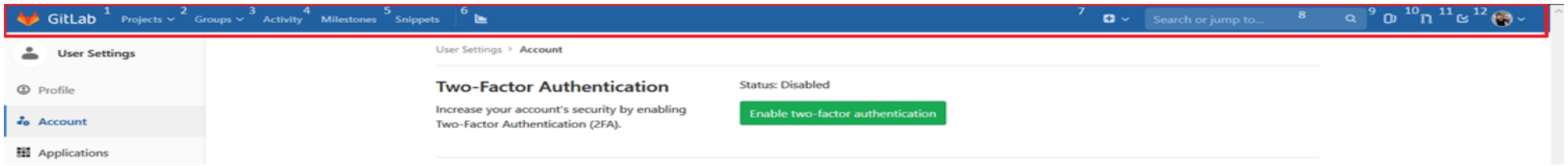
1) Выдача другому пользователю прямого доступа на запись (push) в git-репозитории. Вы можете добавить пользователя в проект в разделе “Участники” (“Members”) настроек проекта, указав уровень доступа. Получая уровень доступа “Разработчик” (“Developer”) или выше, пользователь может отсылать свои коммиты и ветки непосредственно в репозиторий.

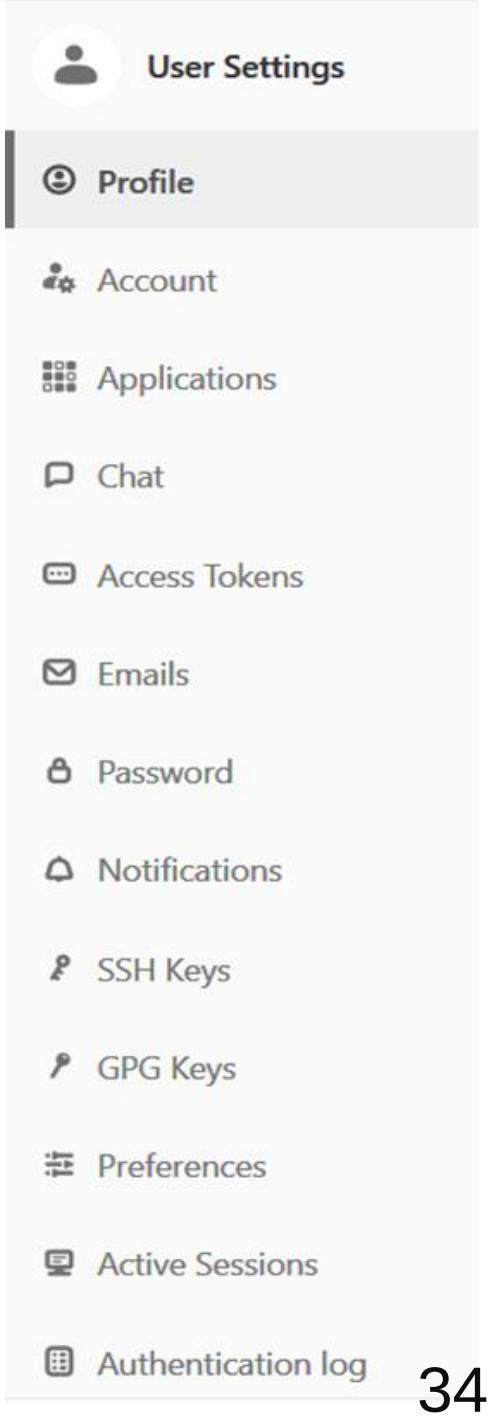
2) Другой, также распространённый способ совместной работы — использование запросов на слияние (merge requests, pull requests). Эта возможность позволяет любому пользователю, который видит проект, вносить свой вклад подконтрольным способом. Пользователи с прямым доступом могут просто создать ветку, отослать в неё коммиты и открыть запрос на слияние из их ветки обратно в master или любую другую ветку.

СТАРТОВОЕ ОКНО GIT LAB



ОКНА НАСТРОЕК GIT LAB

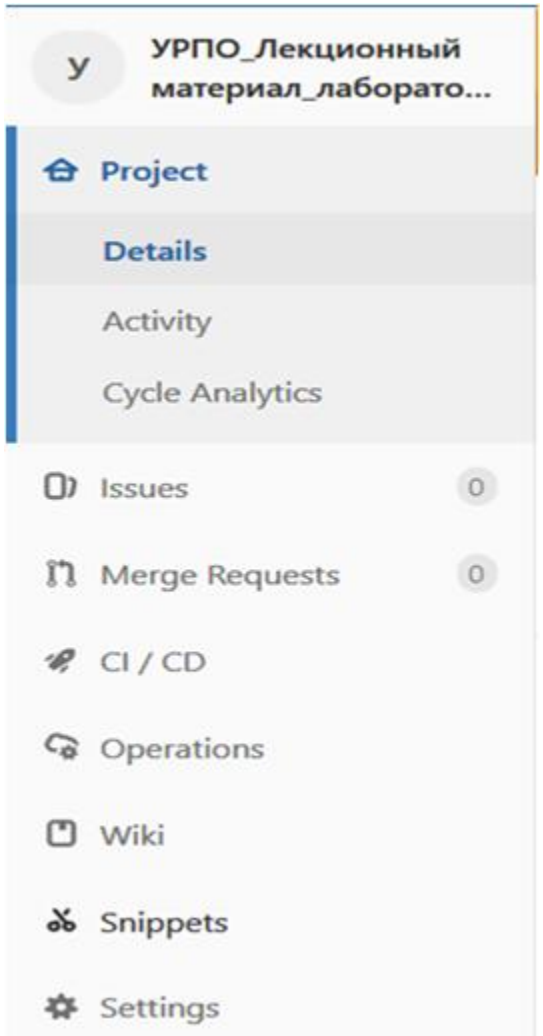




НАСТРОЙКИ ПРОЕКТА

- «Project» - детали проекта. Здесь можно настроить URL проекта, оповещения о действиях на вашу почту. Во вкладке «Activity» вы можете увидеть коммиты, запросы на изменение (pull requests), задачи проекта, баги, комментарии. Во вкладке «Issues» находятся открытые и закрытые задачи и баги;
- «Merge Requests» – запросы на изменение. Текущие и уже выполненные;
- «CI/CD» – Continuous Integration/Continuous Deployment – о нём будет рассказано позднее в нашем курсе, в двух словах можно определить CI и CD как набор инструментов для автоматизации развёртывания приложения и установки его на машину;
- «Operations» – настройки CI/CD;
- «Wiki» – способ написания документации к вашему проекту;
- «Snippets» – позволяет сохранить нужные фрагменты кода в публичном или приватном доступе, если Snippet открыт для других пользователей, есть возможность комментирования;
- «Settings» – различные настройки интеграции, настройки участников команды, различные разрешения, экспортирование проекта.

ОКНО НАСТРОЕК ПРОЕКТА

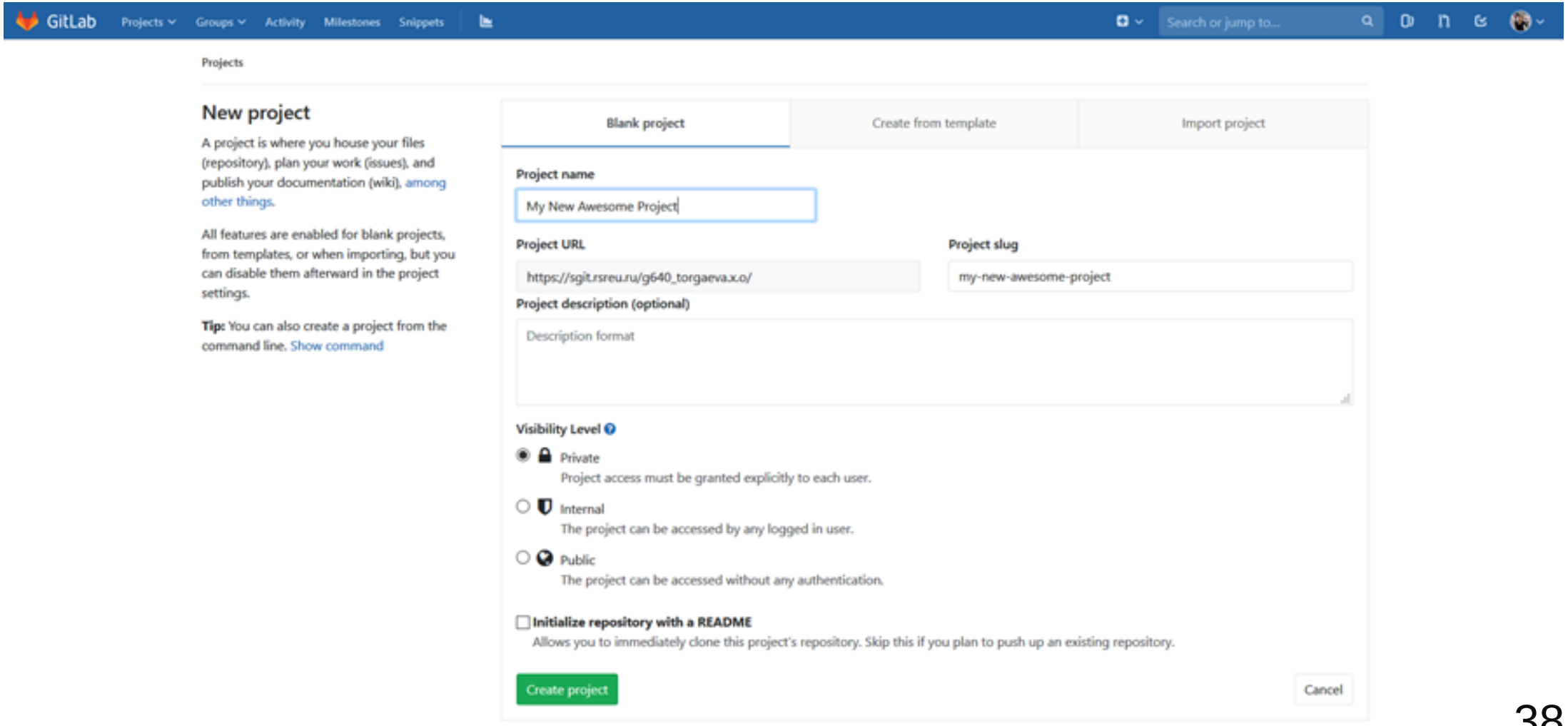


СОЗДАНИЕ ПРОЕТКА В GITLBA

1. 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.

2. 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 помьщью команды git clone <url>.

ПРИМЕР СОЗДАНИЯ ПРОЕКТА



ДОБАВЛЕНИЕ В ГРУППУ ДЛЯ РАБОТЫ С ПРОЕКТОМ

В лабораторной работе вам понадобится добавить человека в группу для работы с одним проектом (make a collaboration). Для этого необходимо зайти в “Settings”->”Members”, на этой странице осуществляется добавление другого человека в участники проекта. Выберите имя участника по его никнейму и права (для лабораторной необходимо установить права на запись (права “developer”)).

FORK ПРОЕКТА

Можно воспринимать Fork проекта как копию чужого репозитория у себя.

Чтобы сделать форк в GitLab, нужно просто перейти на страницу с проектом, и нажать «Fork».

GITHUB

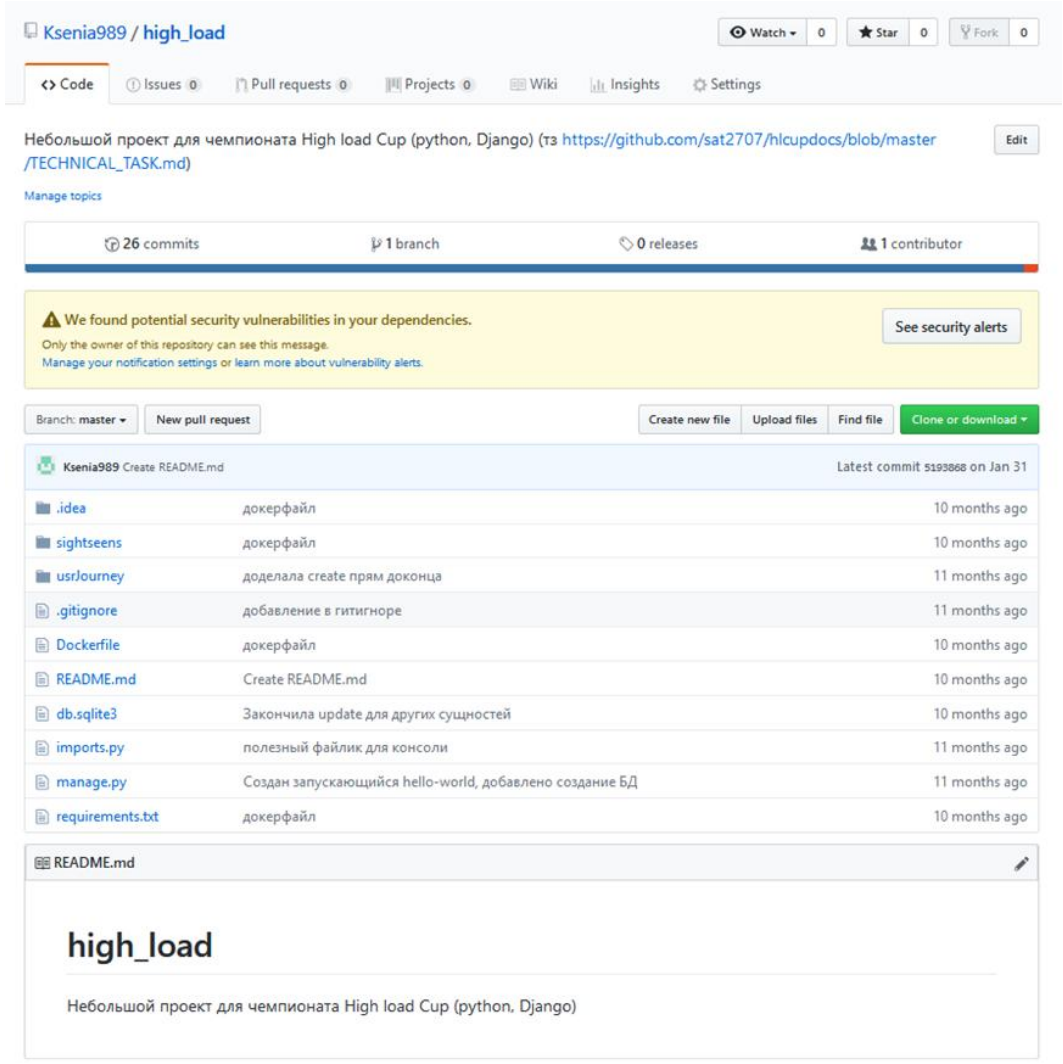
Часто возникающие ситуации, когда вам нужно использовать GitHub:

1) создать свой проект с открытым исходным кодом / использовать как хранилище кода, который не хотелось бы потерять;

2) воспользоваться сторонней библиотекой, являющейся нестандартной;

3) внести вклад в уже существующий проект на GitHub.

ВИД ПРОЕКТА В GITHUB



ИСПОЛЬЗОВАНИЕ СТОРОННИХ БИБЛИОТЕК

- В первую очередь, на то, подходит ли вам этот продукт. Рассмотрите документацию, удовлетворяет ли данное приложение или библиотека целям, для которых вы хотите её использовать;
- Лицензия. Если вам подошёл продукт, посмотите под какой лицензией он распростроняется. Лицензия может предполагать включение информации об авторе в исходный код вашей программы, или не предполагать оного. Так же автор может нести ответственность за поломку косвенный образом из – за ошибки в библиотеке (но чаще всего не несёт).
- Если проект подоходит вам по функциональности и типу лицензии, то смело нажимайте «Clone or download», получайте репозиторий и используйте.

ВНЕСЕНИЯ ВКЛАДА В СУЩЕСТВУЮЩИЙ ПРОЕКТ

Если вам понравилось приложение / библиотека, и вы хотите улучшить её, или вы нашли в ней уязвимость, которую, как вам кажется, вы могли бы исправить, вы можете помочь проекту с открытым исходым кодом и внести свой вклад.

Вы можете склонировать себе код, внести изменения в новой ветви, тщательно протестировать и сделать запрос на слияние.

Зачастую у подобных проектов весьма ограниченное (или отсутствующее) финансирование, и любой будет рад подобному вкладу.

ТЕМЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ

- Модель ветвления git flow
- Поиск ломающего коммита методом половинного деления