

Введение в разработку программного обеспечения (ИС и ЦД)
Системы контроля версий

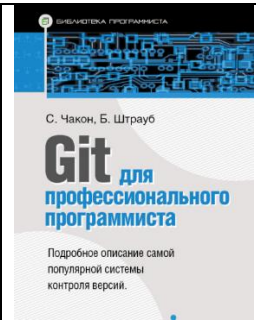
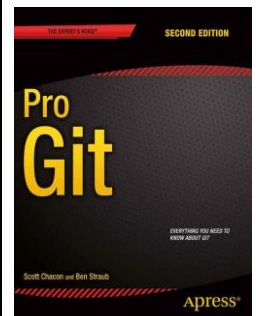
План лекции:

- назначение, разновидности систем контроля версий;
- система контроля версий Git;
- установка и настройка Git;
- три состояния файлов в Git;
- основные команды в Git;
- ветвления в Git;
- создание, слияние веток в Git;
- конфликты при слиянии веток в Git.
- основы работы с удаленными репозиториями;
- совместная работа над проектом (коллаборация).
-

1. Системы контроля версий: назначение и разновидности

Система управления **версиями** (от англ. *VersionControl System*, *VCS* или *Revision Control System*, *RCS*) – программное обеспечение для облегчения работы с изменяющейся информацией и разработки проекта совместно с коллегами.

Литература:

	Выложена на diskstation.belstu.by
	Доступна по ссылке: https://git-scm.com/book/ru/v2

Назначение

Назначение систем контроля версий	<ul style="list-style-type: none">✓ <i>автоматическое создание архива (бэкап) для синхронизации кодовой базы;</i>✓ <i>отслеживание изменений (кто, когда и зачем сделал изменения);</i>✓ <i>совместная работа над одним и тем же проектом;</i>✓ <i>отслеживание ошибок (Bug трекинг-система).</i>
--	--

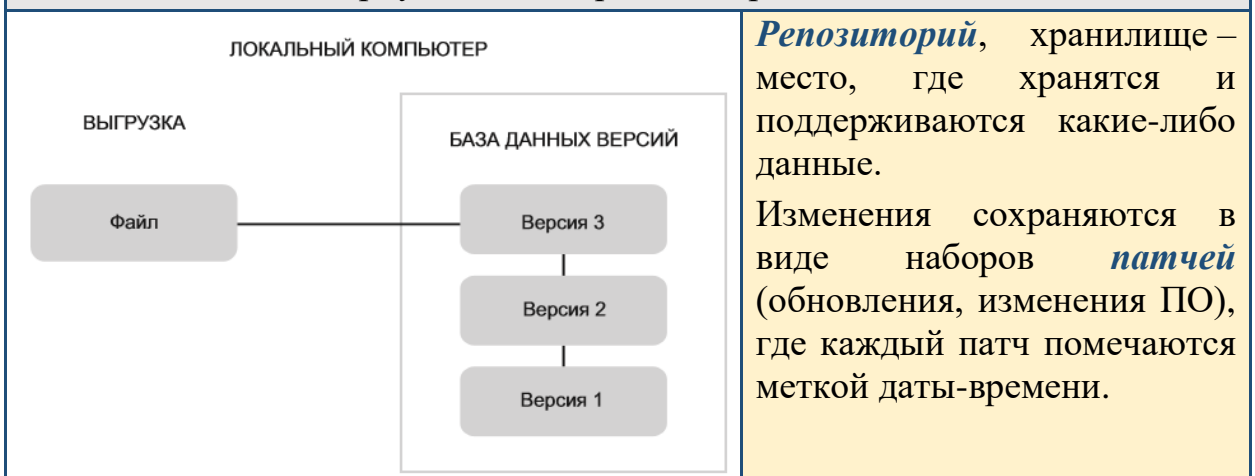
Разновидности

Разновидности систем контроля версий	<ul style="list-style-type: none">✓ <i>локальные системы контроля версий (Version Control System, VCS, Revision Control System, RCS);</i>✓ <i>централизованные системы контроля версий (Centralized Version Control System, CVCS);</i>✓ <i>распределенные системы контроля версий (Distributed Version Control System, DVCS).</i>
---	---

Локальные системы контроля версий

Revision Control System, RCS

записывает историю изменения файла или набора файлов, чтобы в будущем была возможность вернуться к конкретной версии.



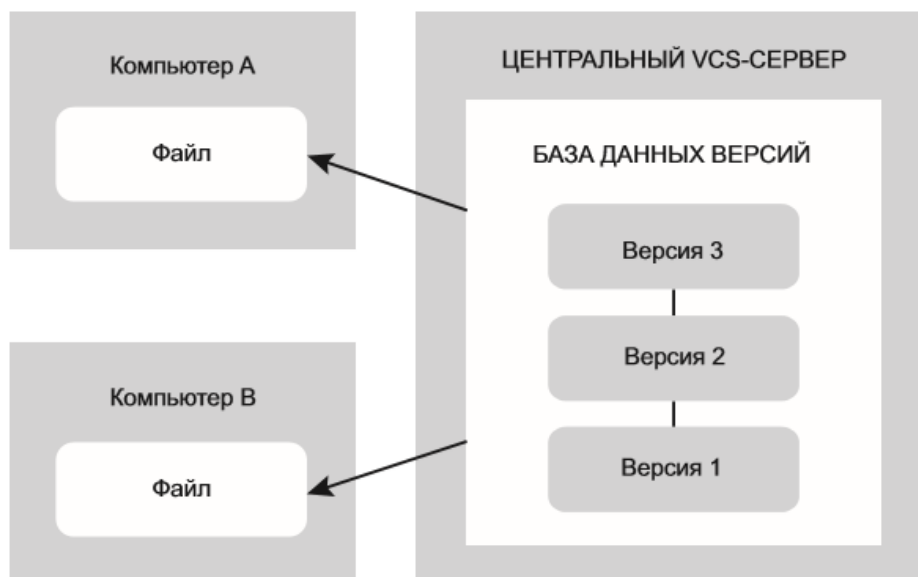
Централизованные системы контроля версий

Centralized Version Control System, CVCS

с единым сервером, содержащим все версии файлов, и набором клиентов, получающих файлы с сервера, что позволяет решить проблему взаимодействия с другими разработчиками.



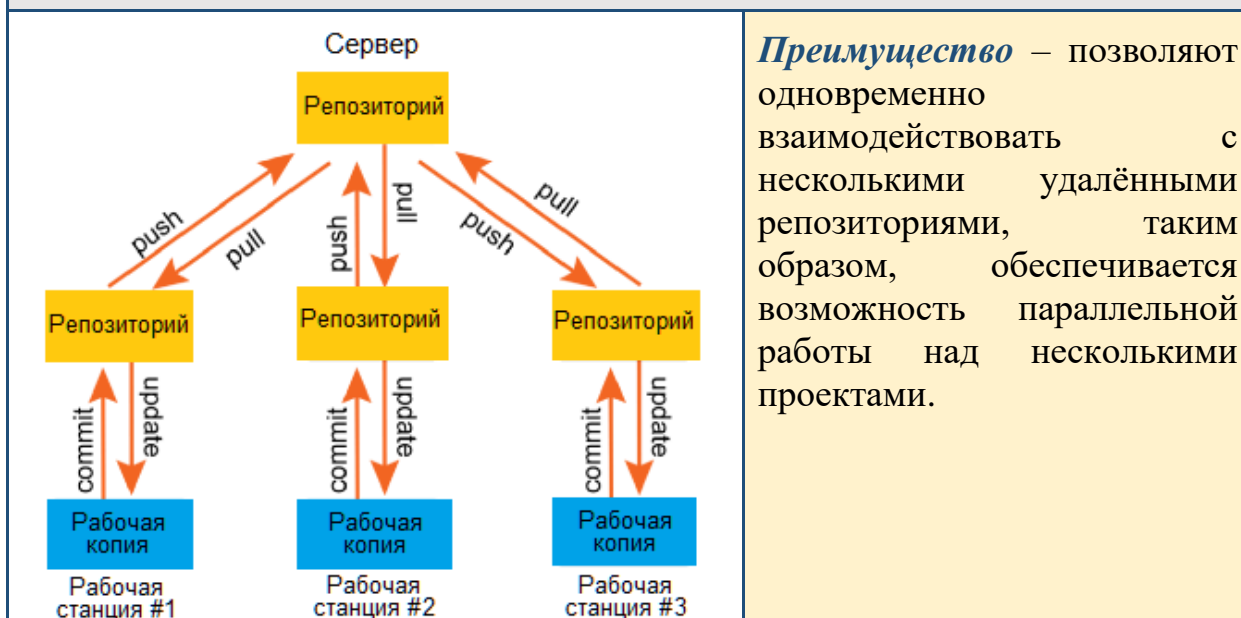
Схема централизованной системы контроля версий:



Распределенные системы контроля версий

Distributed Version Control System, DVCS

клиенты полностью копируют репозиторий (у каждого клиента есть копия всего исходного кода и внесённых изменений).



Сравнение решений

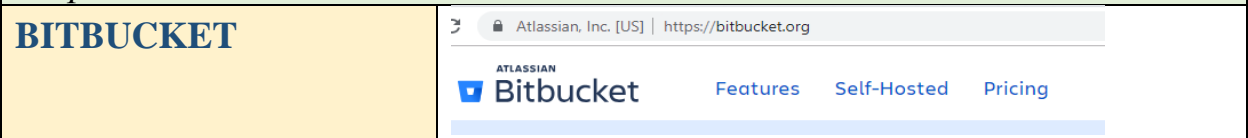
Система одновременных версий (CVS) (80е)	централизованная система управления версиями, популярная в 1990-е – начале 2000-х годов. Хранит историю изменений определённого набора файлов. Создана Диком Груном в 1986 г
SVN (Subversion)	
свободная централизованная система управления версиями. Дата запуска: 2000 г. Язык программирования: Си	
Git	
распределённая система управления версиями. Дата запуска: 2005 г. Язык программирования: Си, командная оболочка UNIX, Perl, Tcl, Python и C++ Разработчик: Линус Торвалдс	
Mercurial	

кроссплатформенная распределённая система управления версиями, разработанная для эффективной работы с очень большими репозиториями кода. Является консольной программой.

Дата запуска: 2005 г.

Язык программирования: Python, Си, Rust

Разработчик: Мэттом Макколлом

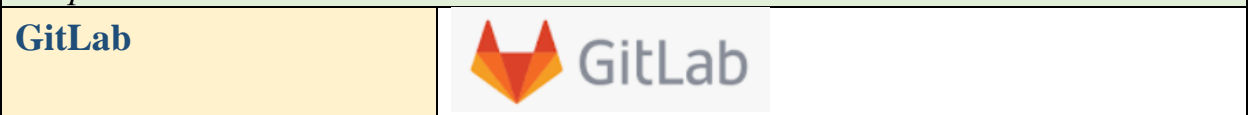


веб-сервис для хостинга проектов и их совместной разработки, основанный на системах контроля версий Mercurial и Git

Дата запуска: 2008 г.

Язык программирования: Python

Разработчик:



веб-приложение с открытым исходным кодом, представляющий систему управления репозиториями кода для Git с собственной вики, системой отслеживания ошибок

Дата запуска: 2011 г.

Язык программирования: Ruby, Go

Система одновременных версий (CVS)

Преимущества:

- ✓ Испытанная временем технология, которая удерживается на рынке десятки лет.

Недостатки:

- ✓ Переименование или перемещение файлов не отражается в истории
- ✓ Риски безопасности, связанные с символическими ссылками на файлы
- ✓ Нет поддержки атомарных операций, что может привести к повреждению кода
- ✓ Операции с ветками программного кода дорогостоящие, так как эта система контроля не предназначена для долгосрочных проектов с ветками кода

Преимущества SVN:

- ✓ Система на основе CVS
- ✓ Допускает атомарные операции
- ✓ Операции с ветвлением кода менее затратны
- ✓ Широкий выбор плагинов IDE
- ✓ Не использует пиринговую модель

Недостатки:

- ✓ Сохраняются ошибки, связанные с переименованием файлов и папок
- ✓ Неудовлетворительный набор команд для работы с репозиторием

- ✓ Сравнительно небольшая скорость

Ключевые особенности Git

- поддерживается автономная работа; локальные фиксации изменений могут быть отправлены позже.
- каждое рабочее дерево в Git содержит хранилище с полной историей проекта.
- ни одно хранилище Git не является по своей природе более важным, чем любое другое.
- скорость работы, ветвление делается быстро и легко.

Преимущества Git:

- ✓ Значительное увеличение быстродействия
- ✓ Дешевые операции с ветками кода
- ✓ Полная история разработки доступная оффлайн
- ✓ Распределенная, пиринговая модель

Недостатки:

- ✓ Высокий порог вхождения для тех, кто ранее использовал SVN
- ✓ Ограниченная поддержка Windows (по сравнению с Linux)

Преимущества Mercurial:

- ✓ По сравнению с Git легче в освоении
- ✓ Подробная документация
- ✓ Распределенная модель системы контроля версий

Недостатки:

- ✓ Нет возможности слияния двух родительских веток
- ✓ Использование плагинов, а не скриптов
- ✓ Меньше возможностей для нестандартных решений

Слоган сервиса Bitbucket («ведро битов»):

Bitbucket is the Git solution for professional teams (Bitbucket - решение Git для профессиональных команд)

2. Система контроля версий Git



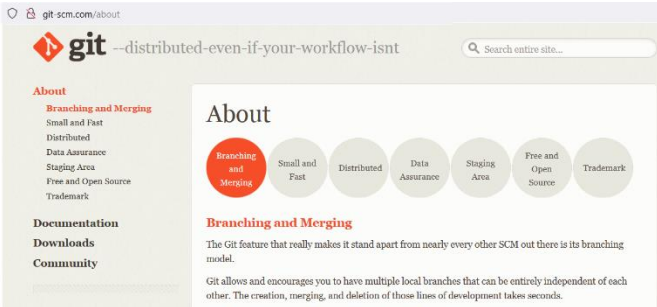
Git - распределённая система управления версиями.

Дата запуска: 2005 г.


Языки программирования:

Си, командная оболочка UNIX, Perl, Tcl, Python и C++

Разработчик: Линус Торвальдс

	<p>Основные свойства:</p> <ul style="list-style-type: none"> ✓ быстроедействие и размер; ✓ безопасность и целостность (хэш SHA); ✓ достоверность; ✓ гибкость (нелинейные рабочие процессы – слияние, ветвление); ✓ производительность (простое ветвление); ✓ функциональность.
---	---

3. Сервис онлайн-хостинга репозитория GitHub

	<p>GitHub — сервис онлайн-хостинга репозитория, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git.</p>
<p>Дополнительно:</p> <ul style="list-style-type: none"> ✓ обучение (глобальный поиск); ✓ реклама (резюме); ✓ контроль доступа; ✓ Bug трекинг; ✓ управление задачами; ✓ вики для каждого проекта. 	

Чем отличается **Git** и **GitHub**

Git:	<ul style="list-style-type: none"> • инструмент, позволяющий реализовать распределённую систему контроля версий.
GitHub:	<ul style="list-style-type: none"> • сервис для проектов, использующих Git.

Что такое Github

Github — крупнейший веб-сервис онлайн-хостинга репозитория, используется для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий **Git**.

Github

- бесплатный сервис для проектов с открытым исходным кодом;
- место сотрудничества миллионов разработчиков и проектов (кроме размещения кода участники могут общаться, комментировать правки друг

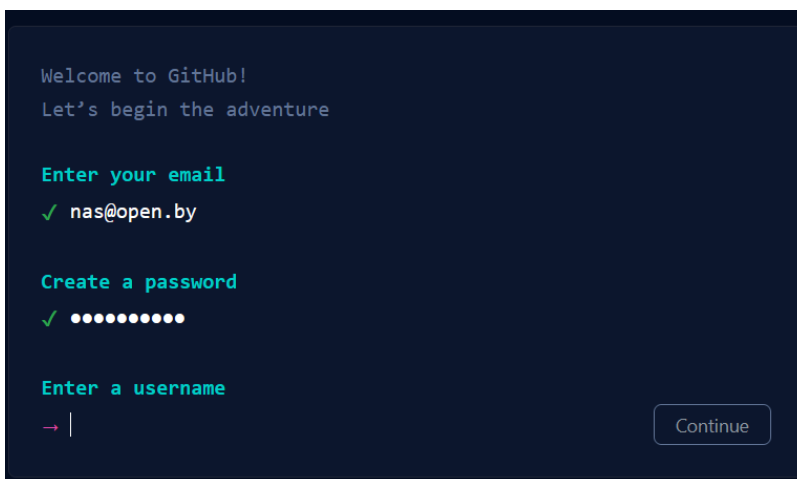
друга, а также следить за новостями знакомых); GitHub – социальная сеть для разработчиков;

- предоставляет удобный интерфейс для совместной разработки проектов и может отображать вклад каждого участника в виде дерева;
- дополнительно для проектов есть личные страницы, вики-разметка и система отслеживания ошибок;
- предоставляет возможность прямо на сайте просмотреть файлы проектов с подсветкой синтаксиса для большинства языков программирования.

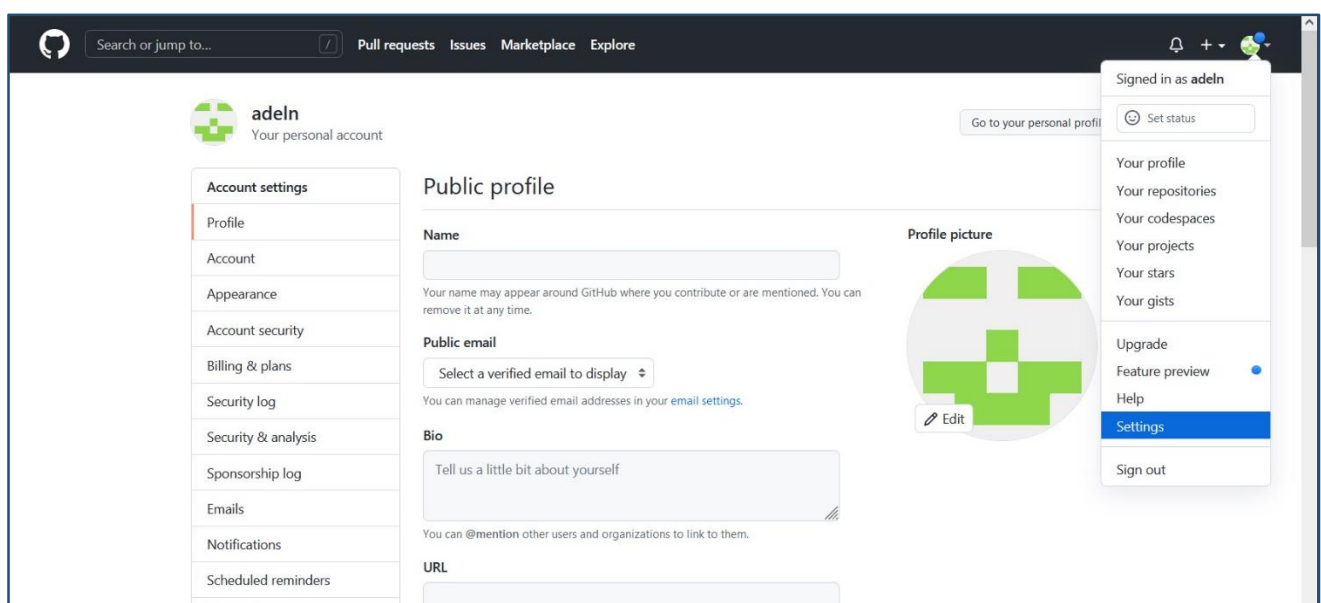
4. GitHub – Сопровождение проекта

Настройка и конфигурация учетной записи

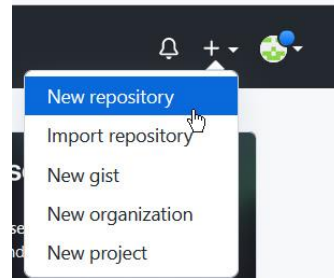
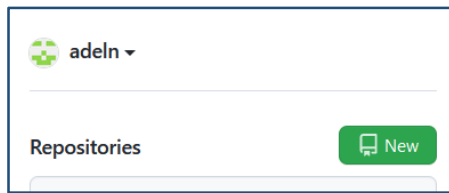
- создать бесплатную учётную запись на сайте <https://github.com/>
- выбрать логин, который еще не занят, указать адрес вашей электронной почты и пароль.



- на следующем шаге (по желанию) можно настроить ваш профиль во вкладке «Settings»

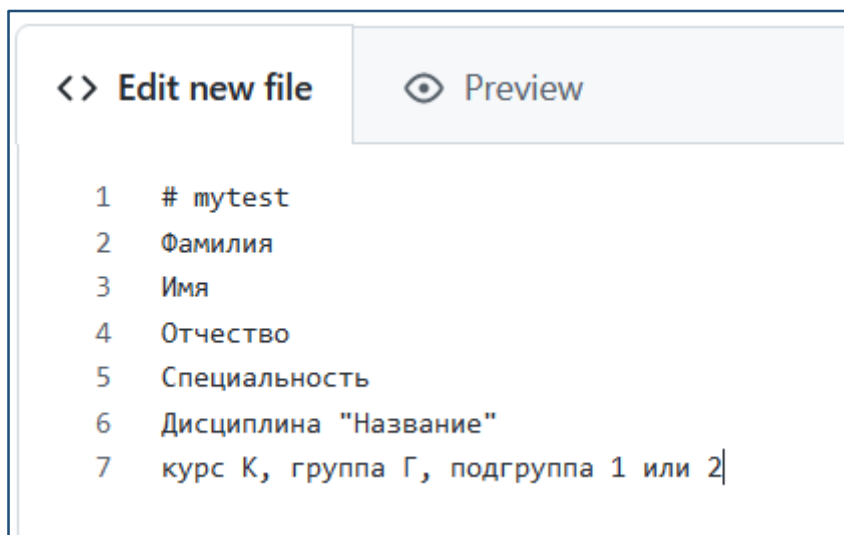
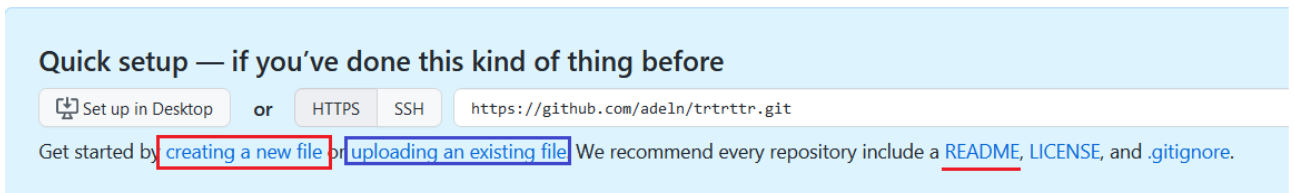


d) Создадим репозиторий с именем mytest (режим доступа Public):

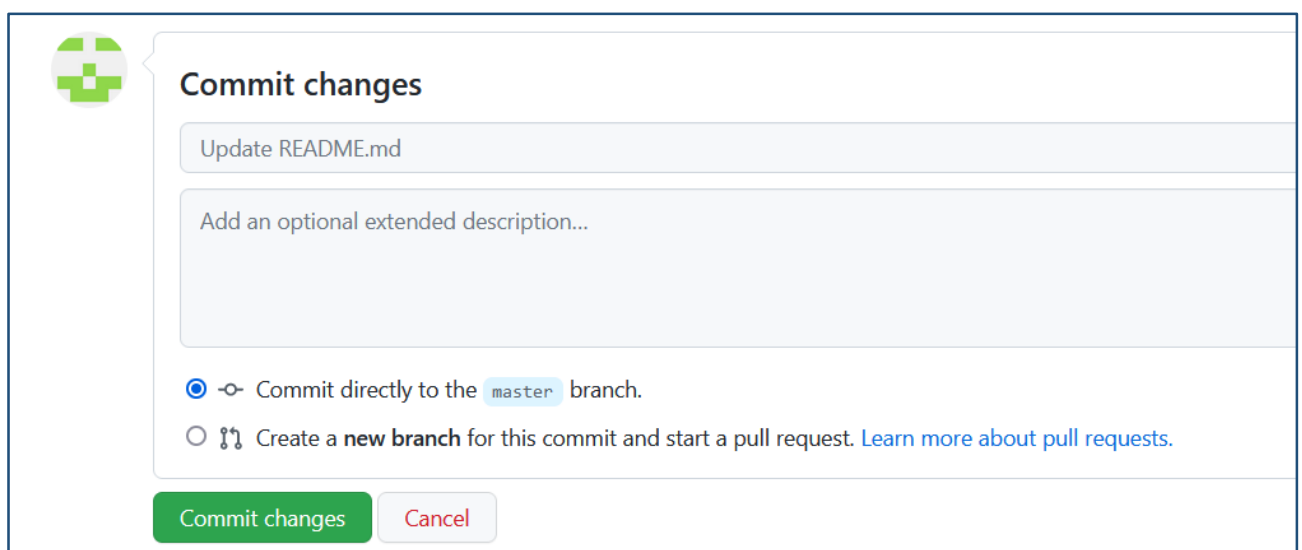


либо из меню

e) Добавим в репозиторий README.md – readme-файл, используя web-интерфейс:



f) Зафиксируем изменения:



g) Можно отследить изменения файла, выбрав этот файл и кликнув по ссылке:

Collaboration.docx	Create issue 1	14 hours ago
README.md	<u>Update README.md</u>	16 hours ago

Update README.md

master

adeln committed 7 days ago Verified

Showing 1 changed file with 1 addition and 0 deletions.

1

README.md

	↑	@@ -2,6 +2,7 @@
2	2	Фамилия
3	3	Имя
4	4	Отчество
5	+	Специальность
5	6	Дисциплина "Название"
6	7	курс К, группа Г, подгруппа 1 или 2
7	8	

5. Установка и настройка Git

Загрузить Git можно с официального сайта: <http://git-scm.com/>

git --everything-is-local

About

Documentation

Downloads

GUI Clients

Logos

Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree

Downloads

macOS

Windows

Linux/Unix

Older releases are available and the [Git source repository](#) is on GitHub.

Latest source Release

2.33.1

[Release Notes \(2021-10-12\)](#)

Download for Windows

Настройка Git

Конфигурирование Git с помощью утилиты *командной строки*

Запустить Git Bash на компьютере.

Глобальными настройками являются *имя пользователя* и его *email*. Их можно установить следующими командами в консоли *Git Bush*:

```
$ git config --global user.name <"Your name">  
$ git config --global user.email <email@example.com>
```

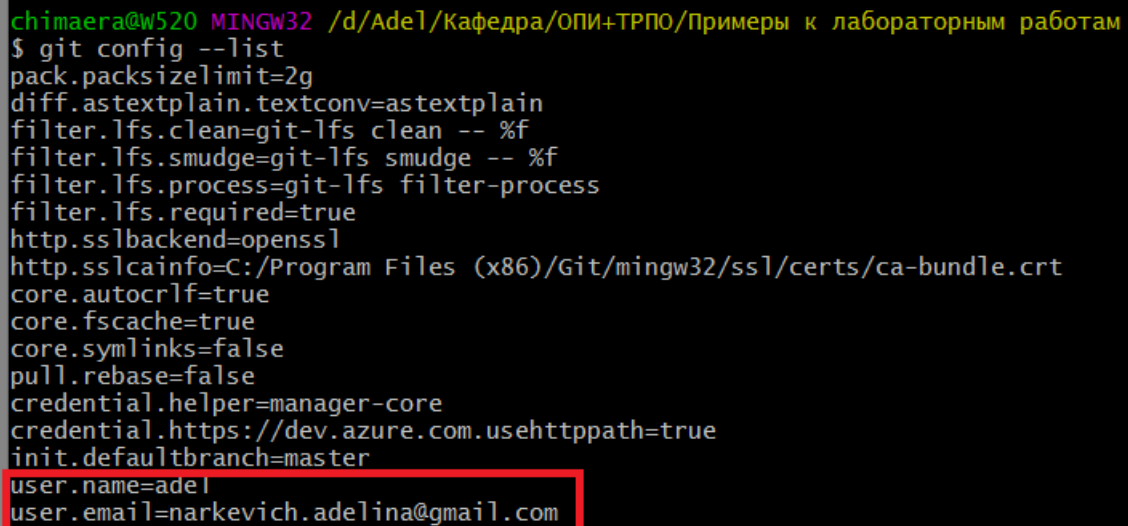
В Git существует три места, где хранятся настройки:

- на уровне системы;
- на уровне пользователя;
- на уровне проекта (репозитория).

Все параметры будут помещены в файл с настройками Git `.gitconfig`, расположенном в домашнем каталоге пользователя

Для просмотра введенных изменений воспользуйтесь командой:

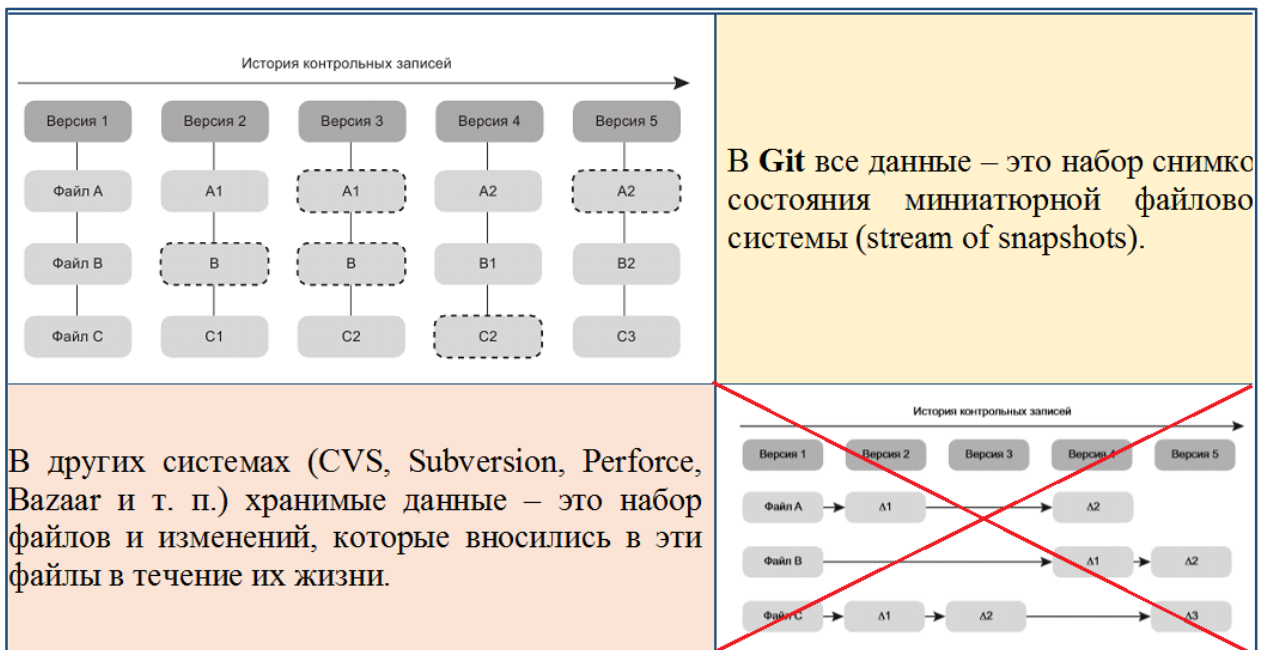
```
$ git config --list
```



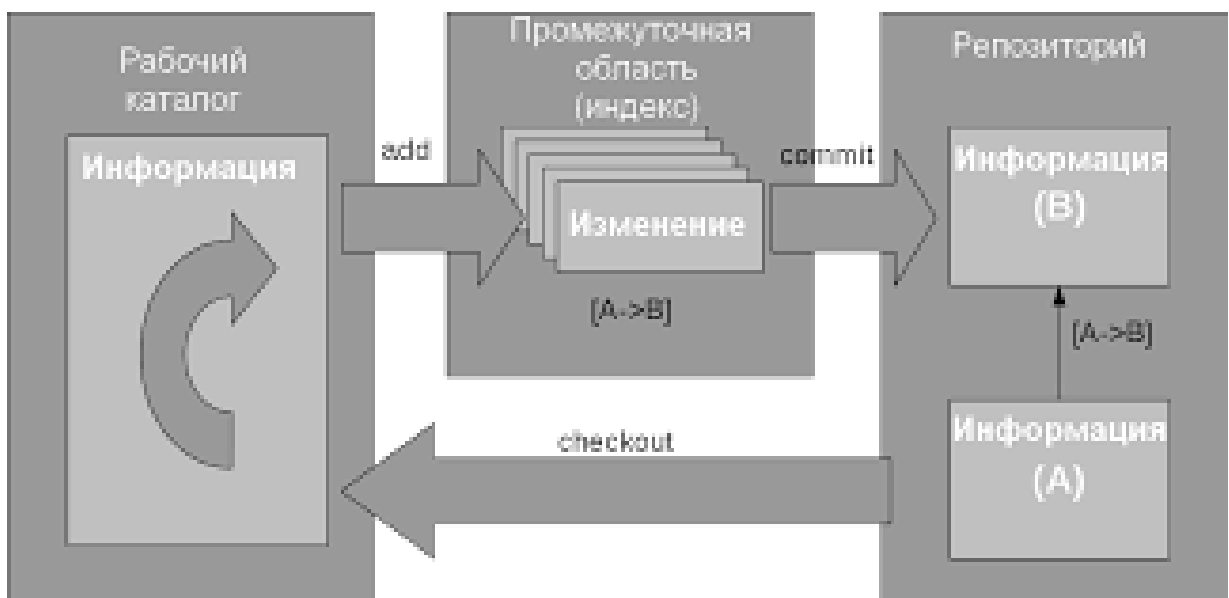
```
chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам  
$ git config --list  
pack.packsizelimit=2g  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
http.sslbackend=openssl  
http.sslcainfo=C:/Program Files (x86)/Git/mingw32/ssl/certs/ca-bundle.crt  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager-core  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=adel  
user.email=narkevich.adelina@gmail.com
```

6. Отличия Git от других систем контроля версий:

1) Хранит снимки состояний, а не изменений



2) Локальность операций



3) Целостность Git (вычисление контрольных сумм – хеш SHA-1)

Хеш - строка из 40 символов, включающая в себя числа в шестнадцатеричной системе (0–9 и a–f) и вычисляемая на основе содержимого файла или структуры папки в Git.

Пример:

2bcad51a4025dde7f4b7c2c28d4f4ac614964475

```

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (m
aster)
$ git log
commit 2bcad51a4025dde7f4b7c2c28d4f4ac614964475 (HEAD -> master)
Author: adel <narkevich.adelina@gmail.com>
Date:   Wed Nov 3 22:45:25 2021 +0300

    added Hello.txt to the repo

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (m
aster)
$ |

```

7. Основные определения

<i>Репозиторий Git:</i>	Git хранит информацию в структуре данных, называемой <i>репозиторий</i> (repository). Репозиторий хранится в папке проекта – в папке <i>.git</i>
<i>Репозиторий хранит:</i>	<ul style="list-style-type: none"> – набор коммитов (<i>commit objects</i>) – набор ссылок на коммиты (<i>heads</i>).
<i>Commit objects</i> содержат:	<ul style="list-style-type: none"> – набор файлов, отображающий состояние проекта в текущий момент времени – ссылки на родительские <i>commit objects</i> – SHA1 имя – 40 символьная строка, которая уникально идентифицирует <i>commit object</i>

Основные команды

git init – создание репозитория
git add <имена файлов> – добавляет файлы в индекс
git commit – выполняет коммит проиндексированных файлов в репозиторий
git status – показывает какие файлы изменились между текущей стадией и HEAD. Файлы разделяются на 3 категории: ***новые*** файлы, ***измененные*** файлы, ***добавленные*** новые файлы
git checkout <SHA1 или метка> – получение указанной версии файла
git push – отправка изменений в удаленный репозиторий
git fetch – получение изменений из удаленного репозитория
git clone <remote url> – клонирование удаленного репозитория себе

Все возможные команды можно получить с помощью команды

```
$ git help
```

8. Фрагмент вывода справки в консоль:

```
$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      [--super-prefix=<path>] [--config-env=<name>=<envvar>]
      <command> [<args>]

These are common Git commands used in various situations:

3 start a working area (see also: git help tutorial)
    clone          Clone a repository into a new directory
    init           Create an empty Git repository or reinitialize an existing
one

work on the current change (see also: git help everyday)
    add            Add file contents to the index
    mv             Move or rename a file, a directory, or a symlink
    restore        Restore working tree files
    rm             Remove files from the working tree and from the index
    sparse-checkout Initialize and modify the sparse-checkout

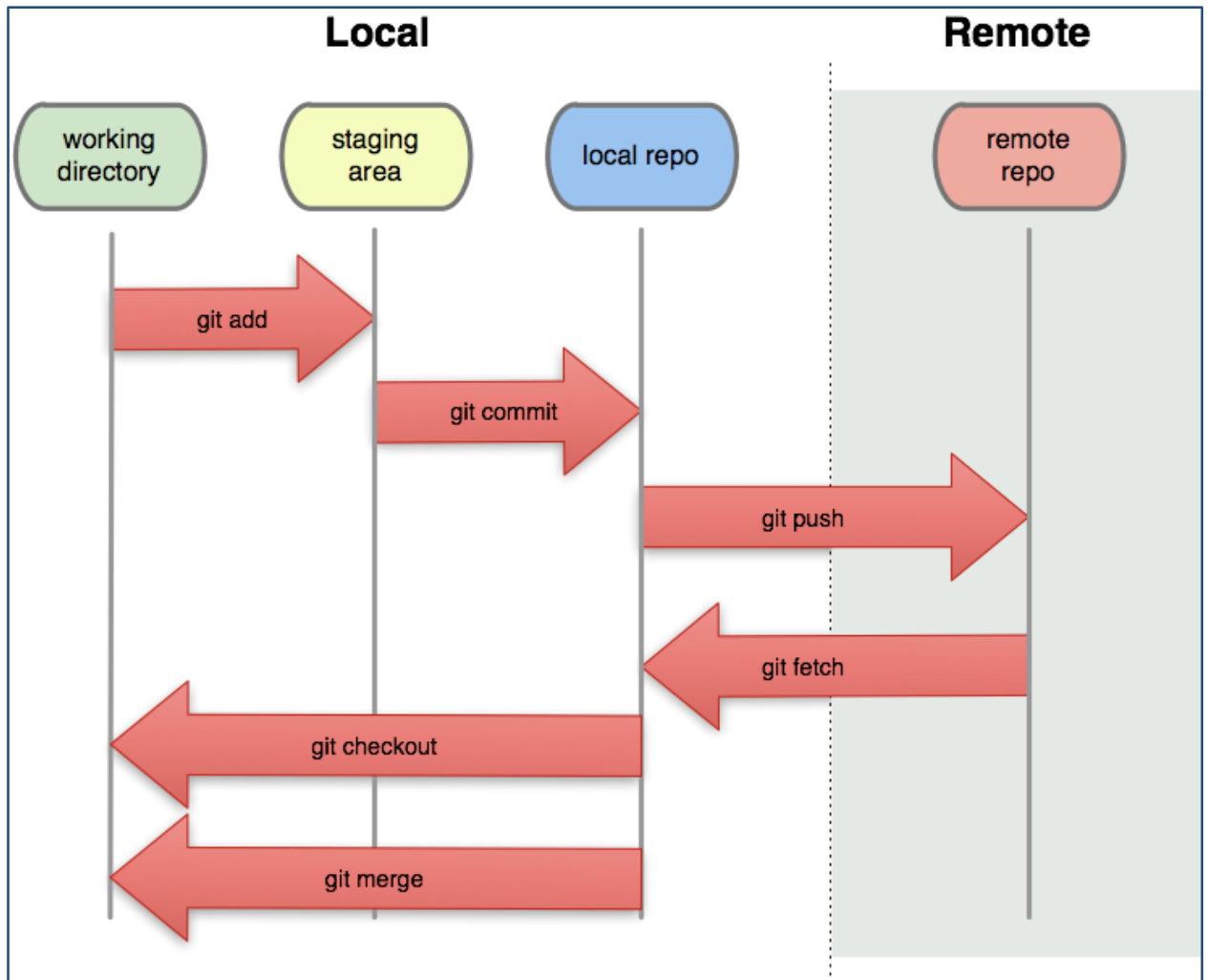
examine the history and state (see also: git help revisions)
```

4) Три состояния файлов



Модифицированное (modified) состояние	изменения уже внесены в файл, но пока не зафиксированы в базе данных
---------------------------------------	--

Индексированное состояние (staged)	текущая версия модифицированного файла помечена как требующая последующей фиксации
Зафиксированное состояние (committed)	данные надежно сохранены в локальной базе



commit



reset



9. Пример создания локального репозитория:

- Перейти в проводнике в рабочую папку, где планируется создать репозиторий, и запустить Git Bash с помощью контекстного меню: «Git Bash Here».
- Инициализация репозитория в выбранной папке выполняется командой

```
$ git init
```

```
chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/К лабораторным
$ git init
Initialized empty Git repository in D:/Adel/Кафедра/ОПИ+ТРПО/К лабораторным/.git/
chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/К лабораторным (master)
$ |
```

В папке появилась новая скрытая папка: **.git** – локальный репозиторий

ОПИ+ТРПО ▸ Примеры к лабораторным работам			
Имя	Дата изменения	Тип	Размер
.git	03.11.2021 22:45	Папка с файлами	
HelloWorld	13.10.2021 20:41	Папка с файлами	
Lec08	27.10.2021 23:50	Папка с файлами	
Hello.txt	03.11.2021 22:32	Текстовый докум...	1 КБ

Текущее состояние (**status**) репозитория отображается командой:

```
$ git status
```

Кроме того, была создана ветка **master**:

```
chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git status
On branch master

No commits yet

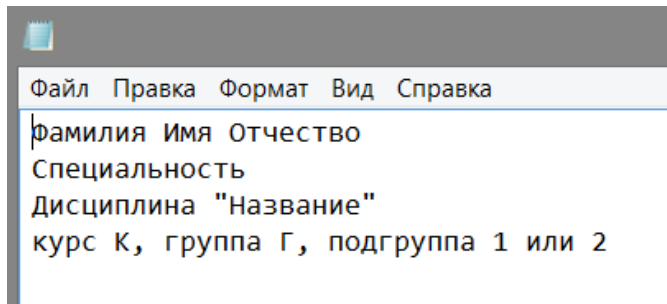
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Hello.txt
    HelloWorld/
    Lec08/

nothing added to commit but untracked files present (use "git add" to track)
```

Красным цветом отмечены новые и модифицированные файлы и папки.

10. Сохранение изменений в репозитории

В папке находится файл `Hello.txt` со следующим содержимым:



Добавим файл `Hello.txt` в репозиторий индексированных файлов командой:

```
$ git add Hello.txt
```

Текущее (обновленное) состояние репозитория отображается командой:

```
$ git status
```

```
$ git add Hello.txt

chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   Hello.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/
```

Теперь файл `Hello.txt` проиндексирован.

Эти изменения можно зафиксировать в репозитории командой:

```
$ git commit -m "added Hello.txt to the repo"
```

Ключ `-m` позволяет добавить комментарий, описывающий, что именно было изменено в коммите ("added Hello.txt to the repo")

```
$ git commit -m "added Hello.txt to the repo"
[master (root-commit) 2bcad51] added Hello.txt to the repo
1 file changed, 4 insertions(+)
create mode 100644 Hello.txt

chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/

nothing added to commit but untracked files present (use "git add" to track)
```

Git информирует об успешном создании нового коммита, в ветку master добавлен 1 файл:

```
[master (root-commit) 2bcad51] added Hello.txt to the repo  
1 file changed, 4 insertions(+)
```

Теперь состояние файла Hello.txt зафиксировано в репозитории.

Все коммиты в Git логируются. Просмотреть журнал можно с помощью команды

```
$ git log
```

которая показывает лог commits начиная с указателя HEAD

```
$ git log  
commit 2bcad51a4025dde7f4b7c2c28d4f4ac614964475 (HEAD -> master)  
Author: adel <narkevich.adelina@gmail.com>  
Date:   Wed Nov 3 22:45:25 2021 +0300  
  
    added Hello.txt to the repo
```

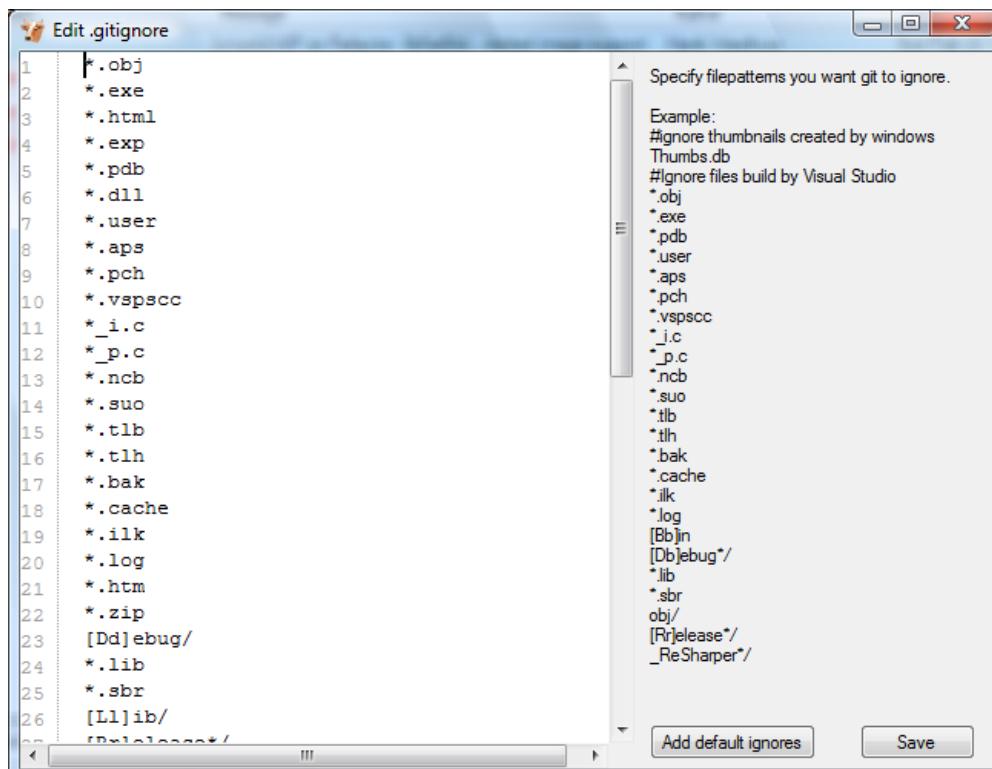
11. Игнорирование файлов

Не все файлы проекта требуется включать в систему контроля версий. Можно исключить:

- настройки IDE;
- результаты сборки;
- файлы кэша;
- индивидуальные файлы пользователя и др.

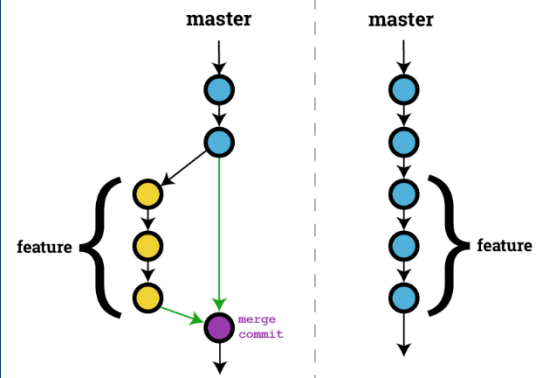
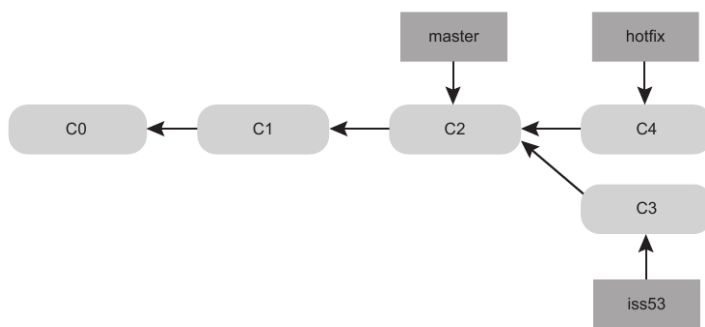
Ознакомиться с шаблонами .gitignore (текстовый файл) можно по ссылке:

<https://github.com/github/gitignore>

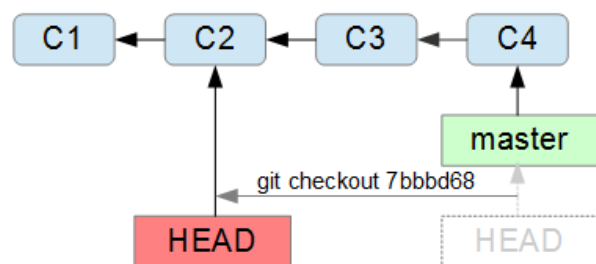
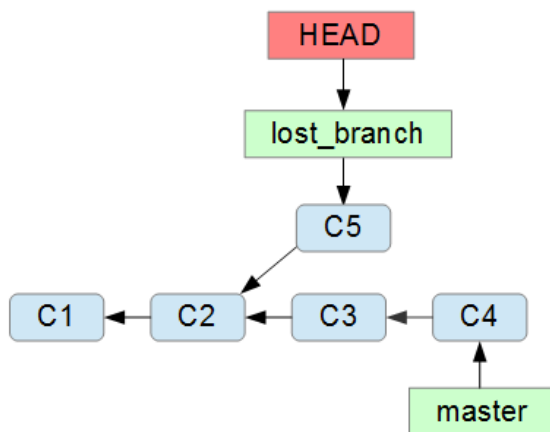


5) Ветвления

Ветвление (branching) означает отклонение от основной линии разработки, после которого работа перестает затрагивать основную линию и переходит в ветвь.



12. Понятие HEAD



Смена веток меняет файлы в рабочей папке

13. Работа с ветками

Создание ветки (branch) выполняется следующей командой:

```
$ git branch <branch_name>
```

```
$ git brabch
git: 'brabch' is not a git command. See 'git --help'.

The most similar command is
  branch
```

Просмотреть список всех веток и определить текущую можно командой:

```
$ git branch
```

```
$ git branch
* master
  test
```

Переключение веток осуществляется командой:

```
$ git checkout <branch_name>
```

```
$ git checkout test
Switched to branch 'test'

chimaera@w520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (test)
$ git branch
  master
* test
```

В результате выполнения команды указатель HEAD сдвигается на ветку test.

Команда переключения веток выполняет 2 функции:

- ✓ сдвигает указатель HEAD на branch_name
- ✓ перезаписывает все файлы в папке на соответствующие новому HEAD

```
$ git status
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/
    Test.txt

nothing added to commit but untracked files present (use "git add" to track)
```

HEAD указывает на ветку test:

```
$ git log
commit 2bcad51a4025dde7f4b7c2c28d4f4ac614964475 (HEAD -> test, master)
Author: adel <narkevich.adelina@gmail.com>
Date:   Wed Nov 3 22:45:25 2021 +0300

    added Hello.txt to the repo
```

Индексируем новый файл и фиксируем изменения в репозитории.

```
$ git status
On branch test
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Test.txt

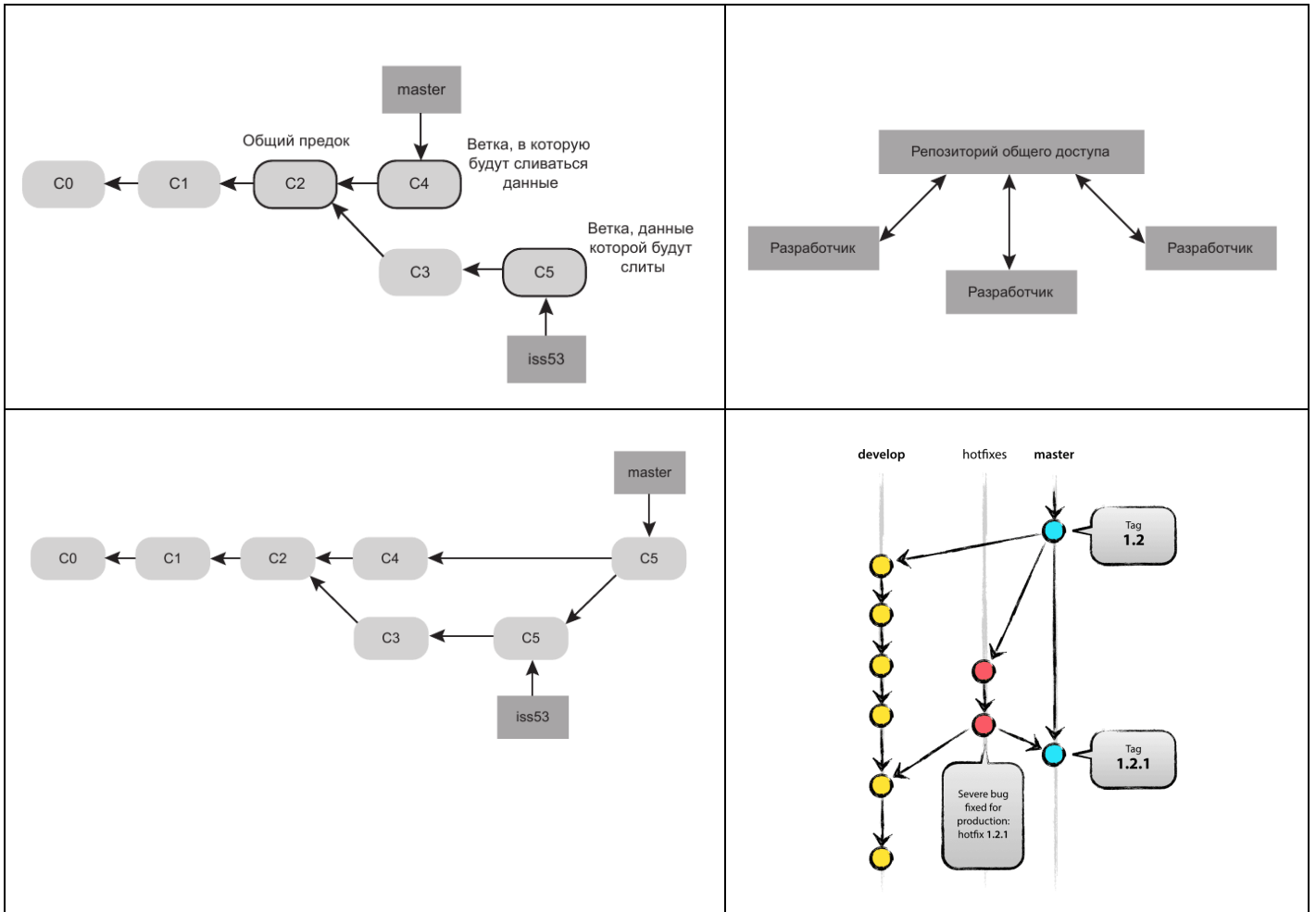
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/

chimaera@w520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (test)
$ git commit -m "added file Test.txt"
[test 621e6/b] added file Test.txt
1 file changed, 1 insertion(+)
create mode 100644 Test.txt

chimaera@w520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (test)
$ git status
On branch test
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/

nothing added to commit but untracked files present (use "git add" to track)
```

6) Слияния веток



Переходим в ветку, в которой будет выполняться слияние (master)

```

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git checkout master
Switched to branch 'master'

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git merge test
Updating 2bcad51..621e67b
Fast-forward
 Test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 Test.txt

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    HelloWorld/
    Lec08/

nothing added to commit but untracked files present (use "git add" to track)

chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам (master)
$ git log
commit 621e67b63da53aa795ae89e1730687252d71ca50 (HEAD -> master, test)
Author: adel <narkevich.adelina@gmail.com>
Date: Thu Nov 4 02:09:56 2021 +0300

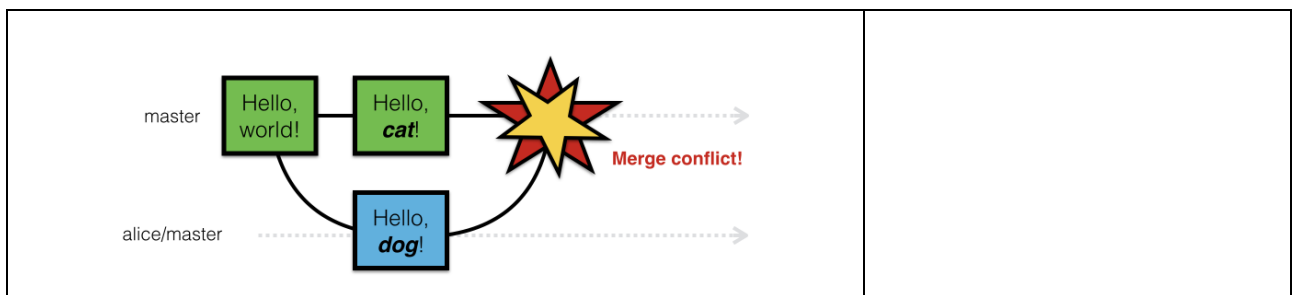
    added file Test.txt

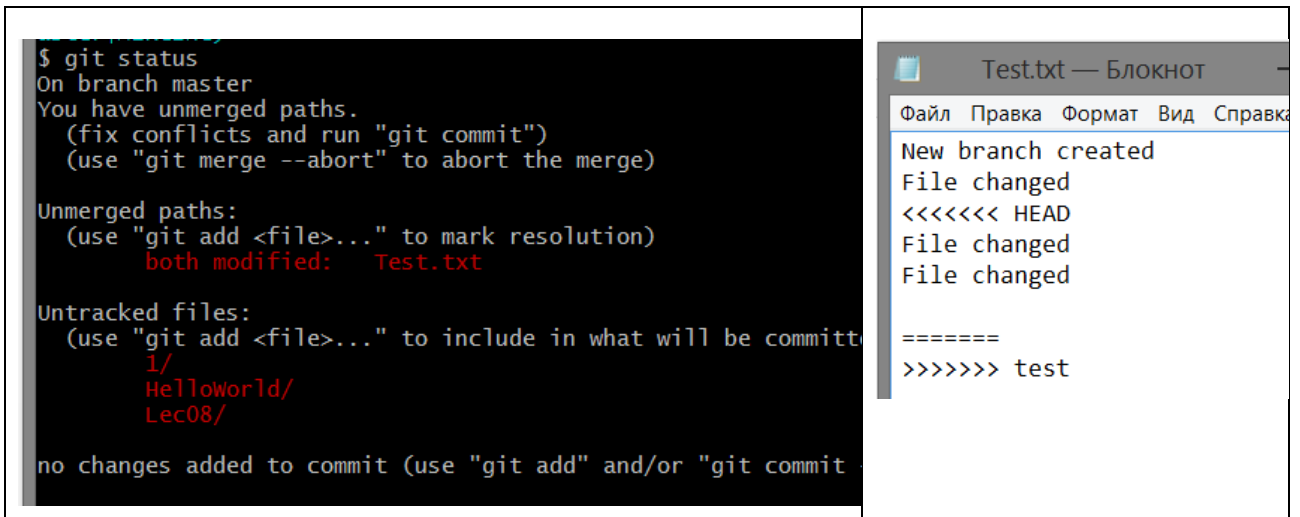
commit 2bcad51a4025dde7f4b7c2c28d4f4ac614964475
Author: adel <narkevich.adelina@gmail.com>
Date: Wed Nov 3 22:45:25 2021 +0300

    added Hello.txt to the repo

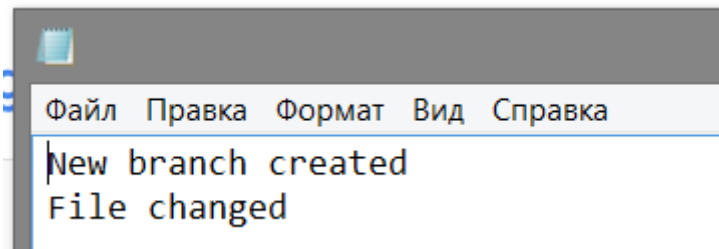
```

Конфликты при слиянии





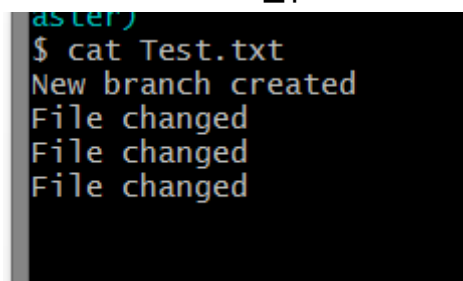
a) Файл `Test.txt` изменен в ветке `test`:



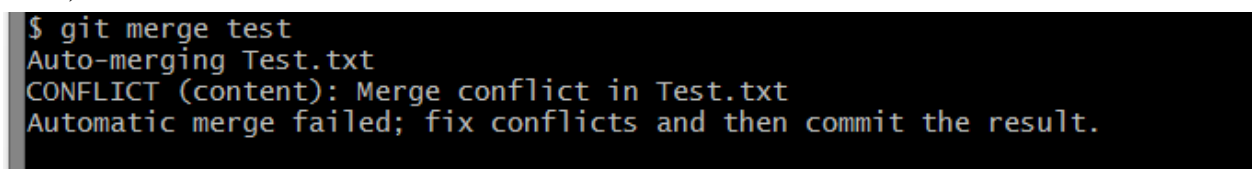
b) Изменения зафиксированы в репозитории.

c) Перешли в ветку `Master` и изменили файл `Test.txt`

d) Присмотр содержимого файла в ветке `master` с помощью команды `cat <имя_файла>`:



e) Пытаемся выполнить слияние веток



Git информирует о конфликте слияния веток. Git не создал коммит слияния автоматически. Процесс остановлен до тех пор, пока вы не разрешите конфликт.

```

$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

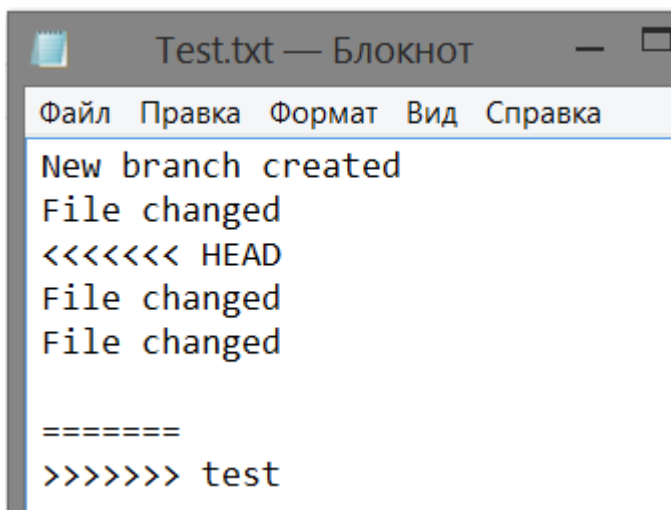
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:   Test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    1/
    HelloWorld/
    Lec08/

no changes added to commit (use "git add" and/or "git commit -a")

```

f) Просмотр файла в папке репозитория:



Показаны различия текста в ветке test и master (помечены несовпадающие места в файле для двух веток:

Начало: <<<<<< HEAD

Конец: =====).

Конфликт разрешается разработчиком вручную.

Клонирование существующего репозитория в Git

Для получения локальной копии существующего Git-репозитория нужно использовать команду `git clone`

Git получает копию практически всех данных, которые есть на сервере.

При выполнении `git clone` с сервера выгружается версия каждого файла из истории проекта.

Т.о., если сервер выйдет из строя, то можно использовать любой из клонов на любом из клиентов, для того, чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования – все данные, помещенные под версионный контроль, будут сохранены

а) Клонирование репозитория в Git осуществляется командой `git clone <url>`:

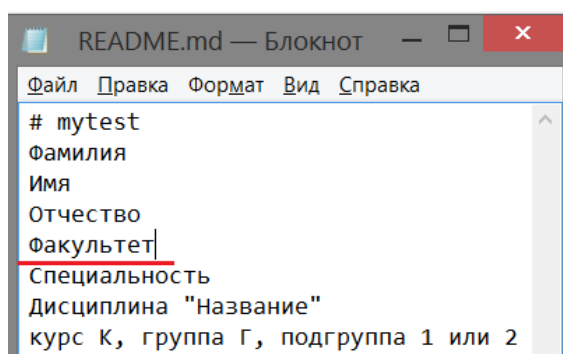
```
$ git clone https://github.com/adeln/mytest
```

```
chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРО/Примеры к лабораторным работам/С1
$ git clone https://github.com/adeln/mytest
Cloning into 'mytest'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 39 (delta 1), reused 0 (delta 0), pack-reused 33
Receiving objects: 100% (39/39), 10.79 KiB | 1.35 MiB/s, done.
Resolving deltas: 100% (6/6), done.
```

Просмотр конфигурационного файла:

```
chimaera@W520 MINGW32 /d/Ade1/Кафедра/ОПИ+ТРО/Примеры к лабораторным работам/С1
$ git config -l
pack.packsizelimit=2g
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files (x86)/Git/mingw32/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager-core
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=master
user.name=adel
user.email=narkevich.adelina@gmail.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
remote.origin.url=https://github.com/adeln/mytest
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
branch.master.remote=origin
branch.master.merge=refs/heads/master
```

б) Внесем изменения в файл README.md



Просмотр содержимого файла README.md в локальной репозитории с помощью программы cat:

```
one/mytest (master)
$ cat README.md
# mytest
Фамилия
Имя
Отчество
Факультет
Специальность
Дисциплина "Название"
курс К, группа Г, подгруппа 1 или 2
```

На скриншоте просмотр *состояния* репозитория, *индексирование* изменений файла README.md и просмотр *текущего состояния* репозитория:

```
one/mytest (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")

chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам/С1
one/mytest (master)
$ git add .

chimaera@W520 MINGW32 /d/Adel/Кафедра/ОПИ+ТРПО/Примеры к лабораторным работам/С1
one/mytest (master)
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   README.md
```

Фиксация изменений в репозиторий:

```
$ git commit -m "Updated from local repo"
[master 8396e75] Updated from local repo
 1 file changed, 2 insertions(+), 1 deletion(-)
```

Просмотр удаленных репозитория в Git

Просмотреть в Git список *настроенных* удаленных репозитория можно командой

```
$ git remote.
```

Выполнение: выведены названия доступных удаленных репозиторий, где origin - имя по умолчанию, которое Git дает серверу, с которого производилось клонирование:

```
one/mytest (master)
$ git remote
origin
```

Просмотр log файла*:

```
$ git log
commit 8396e75fbbfb1530f42ab5a7fa8ebd3f908a5ed7 (HEAD -> master)
Author: adel <narkevich.adelina@gmail.com>
Date: Wed Nov 10 17:51:02 2021 +0300

    Updated from local repo

commit 722fa55b187e36d72eb4dc9fd87b873d153524c5 (origin/master, origin/HEAD)
Author: adel <narkevich.adelina@gmail.com>
Date: Wed Nov 3 22:31:57 2021 +0300

    Update README.md

commit 61abfc0b0826daf0b6915a89a6ed218a96c8f1dd
Author: adel <narkevich.adelina@gmail.com>
Date: Wed Nov 3 14:44:51 2021 +0300

    Update README.md

commit 549b29e35675e42c9cddc13b8860d0e653180260
Author: adel <narkevich.adelina@gmail.com>
Date: Fri Sep 9 18:56:00 2016 +0300
```

Отправка изменений в удаленный репозиторий (Push)

Поделиться своими наработками и отправить их в удаленный репозиторий можно с помощью команды `git push`. Формат команды:

```
git push <имя_удаленного_репозитория> <имя_локальной_ветки>.
```

Выполним отправку изменений в удаленный репозиторий:

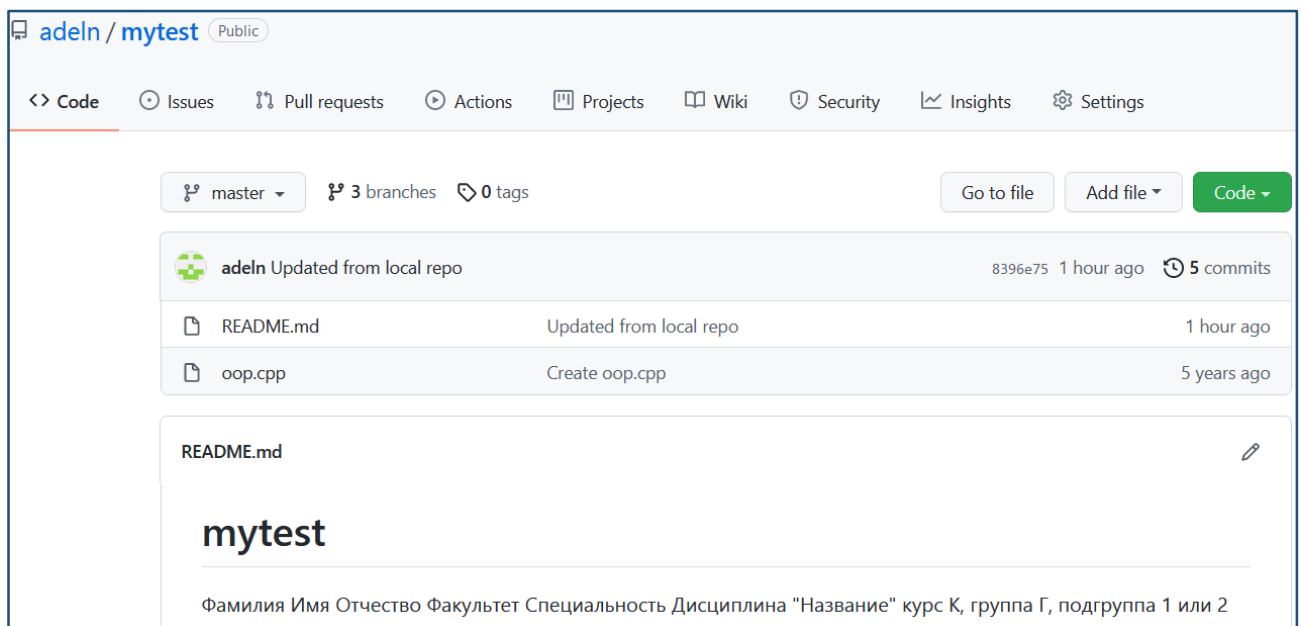
```
$ git push origin master
```

Эта команда выполнится только в случае, если с сервера клонирован репозиторий, к которому у вас есть права на запись (в нашем случае удаленный репозиторий с правами доступа Public)

Результат успешного выполнения команды:

```
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 338 bytes | 338.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/adeln/mytest
722fa55..8396e75 master -> master
```

Присмотр удаленного репозитория:



Указано, что файл `README.md` изменен в локальном репозитории и эти изменения зафиксированы (коммит `8396e75fbbfb1530f42ab5a7fa8ebd3f908a5ed7`, см. лог-файл выше, помеченный *)

Получение изменений из удаленного репозитория — Fetch и Pull

Изменим файл `Readme.md` в удаленном репозитории через web-интерфейс, добавив текущую дату и время.

Зафиксируем изменения и «заберем» изменения из удаленного репозитория в локальный.

формат команды fetch: `git fetch <URL_удаленного_репозитория>`

Выполним команду:

```
$ git fetch https://github.com/adeln/mytest
```

```
$ git fetch https://github.com/adeln/mytest
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 811 bytes | 202.00 KiB/s, done.
From https://github.com/adeln/mytest
* branch                HEAD      -> FETCH_HEAD
```

По команде выполняется обращение к указанному удаленному репозиторию и забираются все те данные проекта, которых у вас ещё нет. При этом их слияния с вашими наработками не происходит и то, над чем вы работаете в данный момент, не модифицируется.

```
$ git show
commit 8396e75fbbfb1530f42ab5a7fa8ebd3f908a5ed7 (HEAD -> master, origin/master,
origin/HEAD)
Author: adel <narkevich.adelina@gmail.com>
Date:   Wed Nov 10 17:51:02 2021 +0300

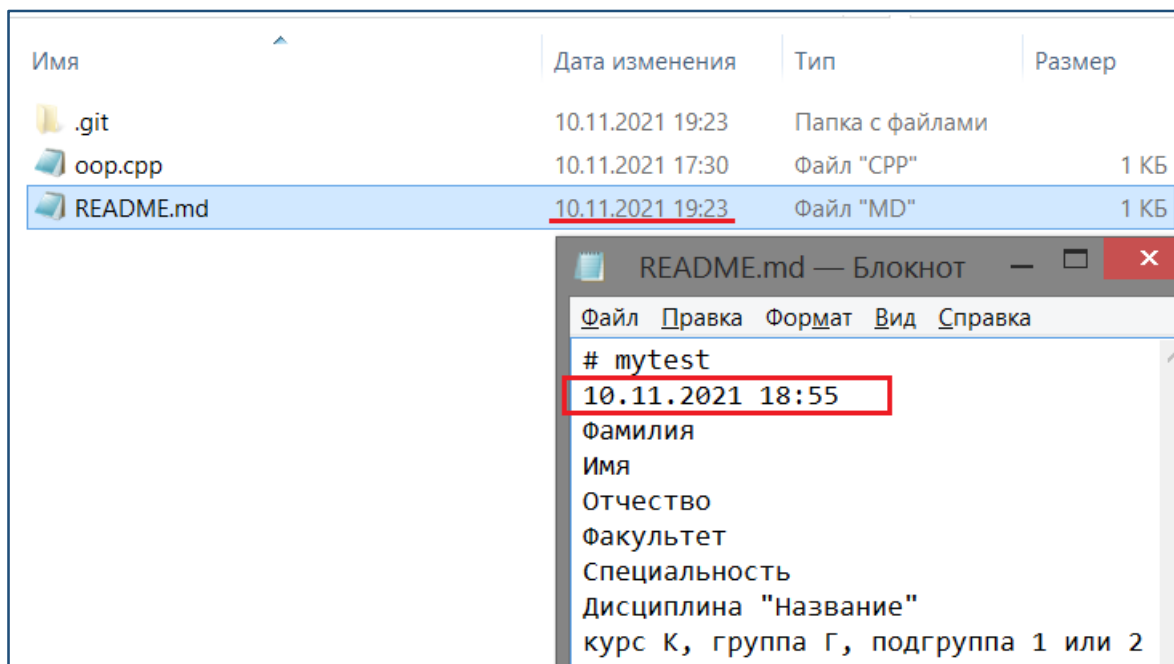
    Updated from local repo

diff --git a/README.md b/README.md
index 2092e47..b61892c 100644
--- a/README.md
+++ b/README.md
@@ -1,7 +1,8 @@
-# mytest
+# mytest
Фамилия
Имя
Отчество
+Факультет
Специальность
Дисциплина "Название"
курс К, группа Г, подгруппа 1 или 2
```

Можно использовать команду `git pull` чтобы автоматически получить изменения из удалённой ветки и слить их со своей текущей:

```
$ git pull https://github.com/adeln/mytest
From https://github.com/adeln/mytest
* branch                HEAD      -> FETCH_HEAD
Updating 8396e75..b0ec98f
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Изменения в папке локального репозитория:



Просмотр лог-файла:

```
$ git log
commit b0ec98f9866f630569e93558425ef0d668d1a725 (HEAD -> master)
Author: adeln <narkevich.adelina@gmail.com>
Date:   Wed Nov 10 18:55:16 2021 +0300

    Update README.md

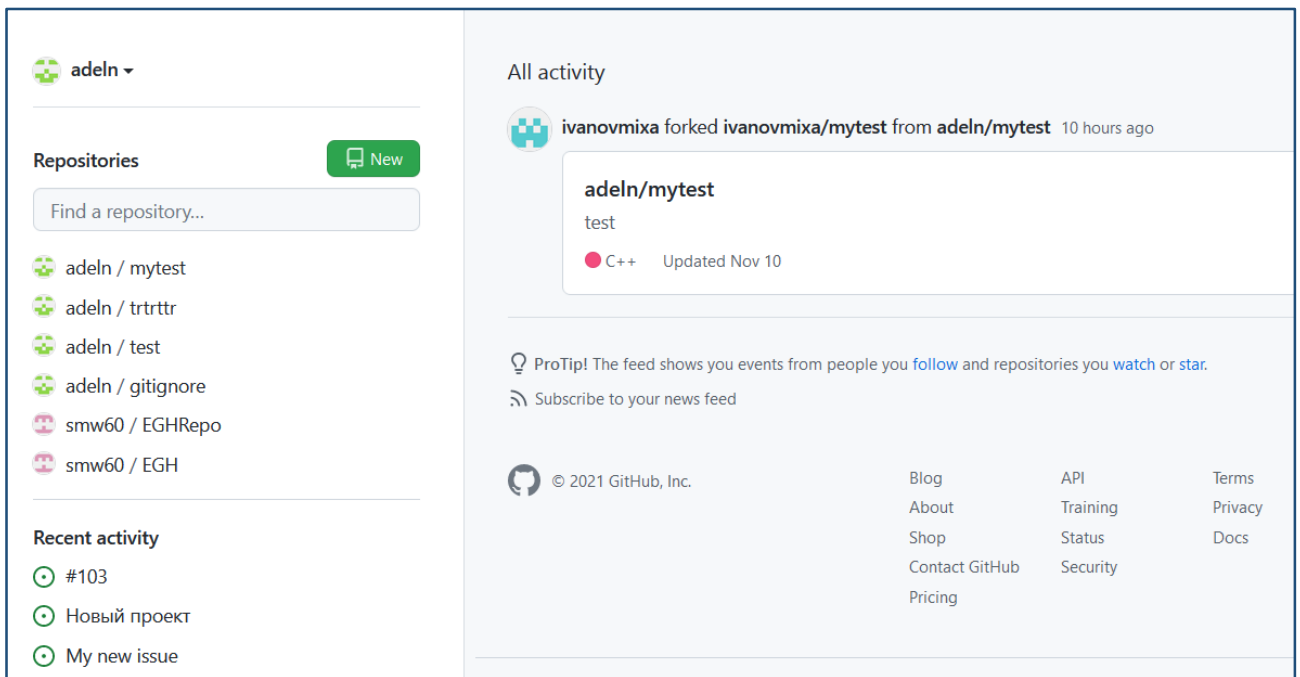
commit 8396e75fbbfb1530f42ab5a7fa8ebd3f908a5ed7 (origin/master, origin/HEAD)
Author: adel <narkevich.adelina@gmail.com>
Date:   Wed Nov 10 17:51:02 2021 +0300

    Updated from local repo
```

14. Совместная работа над проектом

Совместная работа с репозиторием требуется, когда необходимо учитывать текущие задачи, выполнять требования к ним и исправлять баги.

- а. На главной странице аккаунта отображается содержимое и все текущие активности:



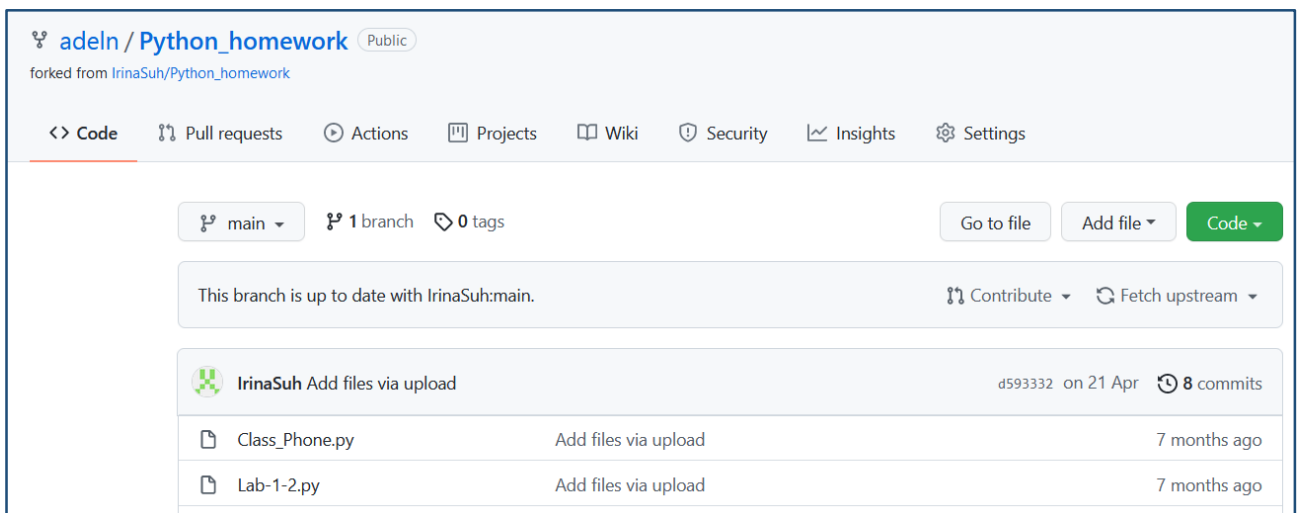
в. Копирование репозитория в Github.

Последовательность действий:

- **создайте форк репозитория коллеги:**

Нажать кнопку  в верхнем правом заголовке проекта коллеги.

Репозиторий коллеги клонирован в отдельную ветку:



- **внесите изменения в своей ветке форка (например, добавив какой-нибудь файл).**
- **создайте запрос (pull-реквест) в репозиторий коллеги, предложив свои изменения:**

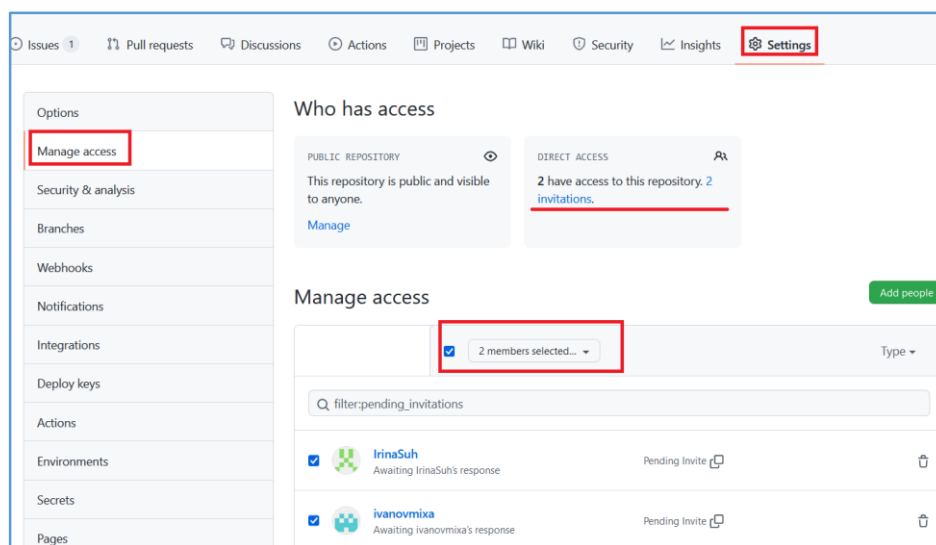
Pull requests

- после слияния ваших изменений в исходный репозиторий его владельцем, заберите в свой форк последние изменения.

с. Добавление членов команды: организация и соавторы

Существует два способа настройки Github для совместной работы:

- **Организации.** Владелец организации может создавать множество команд с разными уровнями доступа для различных репозиториев.
- **Сотрудники.** Владелец репозитория может добавлять коллабораторов с доступом Read + Write для одного репозитория.



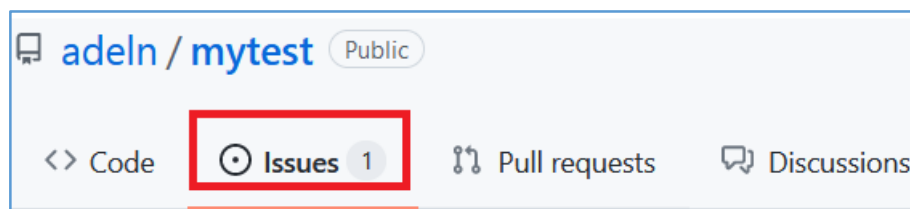
д. Создание проблемы (issue)

Для этого нужно включить вкладку issues.

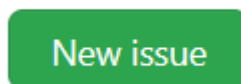
Сделать это можно так: настройка проекта, отметить галочку issues. Появляется вкладка issues, с помощью которой можно ставить задачи и обсуждать их.

Перейти на страницу репозитория.

Под заголовком репозитория выбрать меню Issues:



Нажать кнопку



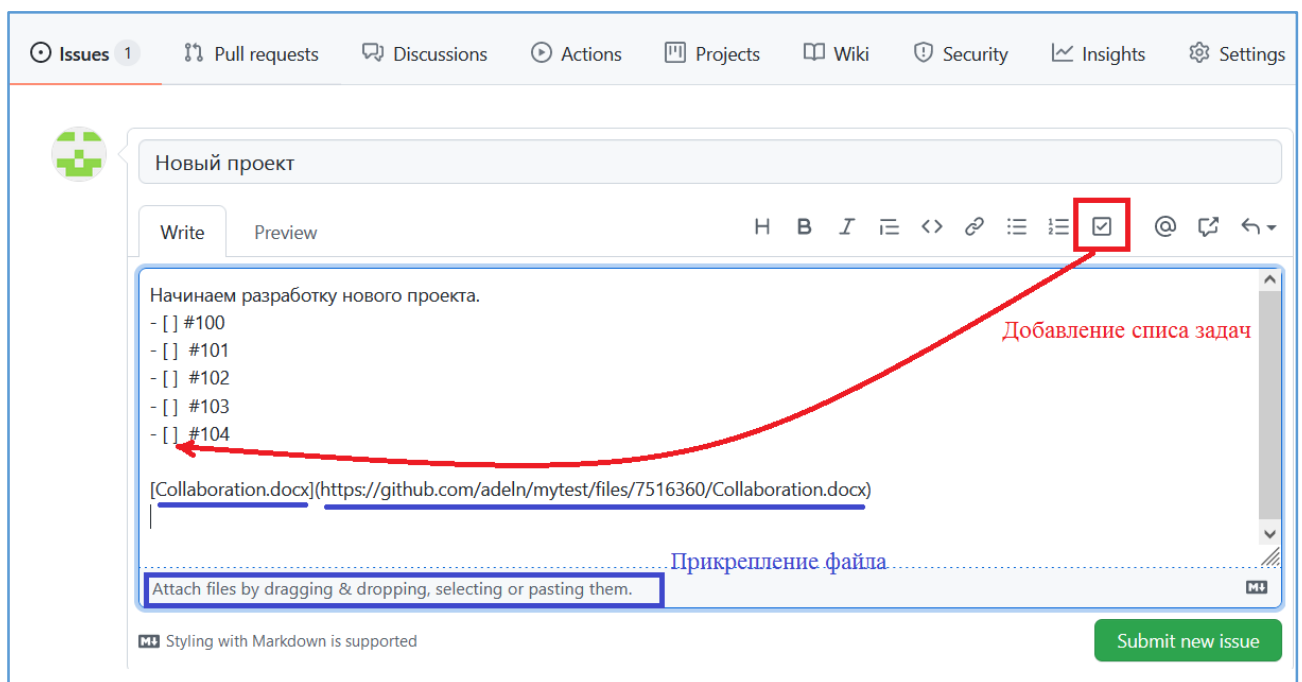
е. Заполнение информации

В поле заголовка **даем** проблеме **название** (название должно отражать суть проблемы).

В закладке Write рабочей области ввода **добавляем** текстовое **описание**, объясняющее цель проблемы, включая любые подробности, которые могут помочь решить проблему. В нашем случае иницилируем начало разработки нового проекта. Определяем цель и ожидаемый результат.

Добавляем список задач – этапы разработки проекта (перед каждым элементом списка надо поставить символ []). Элементы списка могут быть обычным текстом или ссылками на существующие проблемы по их номеру либо по URL. Текст можно отформатировать.

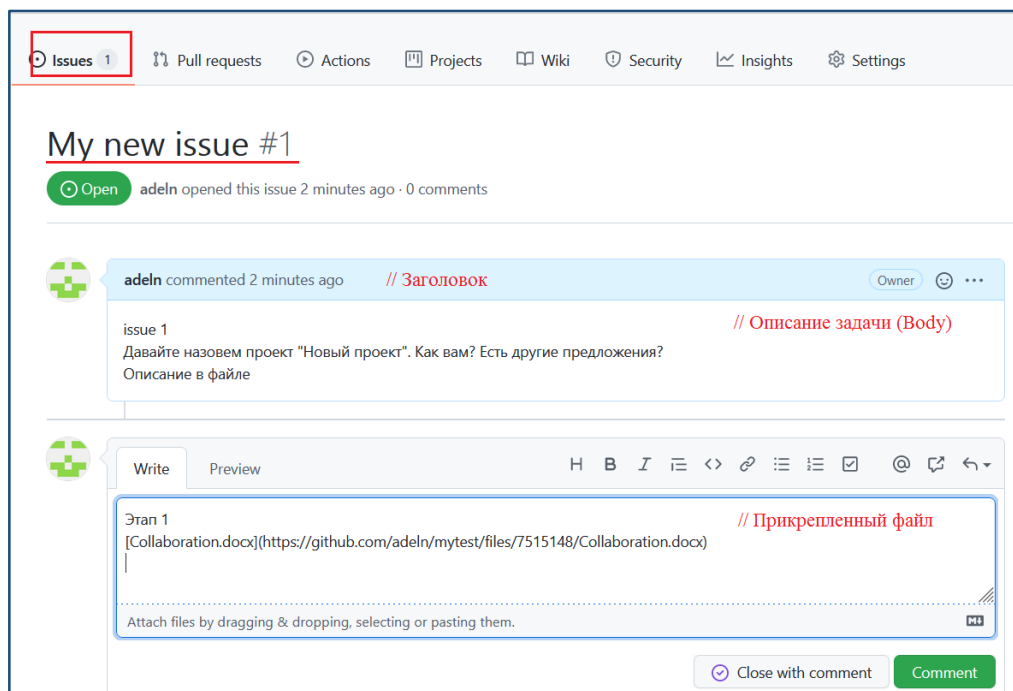
Так же можно прикрепить файл с дополнительной информацией.



Прикрепить файл можно несколькими способами:

- перетащить его из папки;
- выбрать из пункта «выбрать файл»

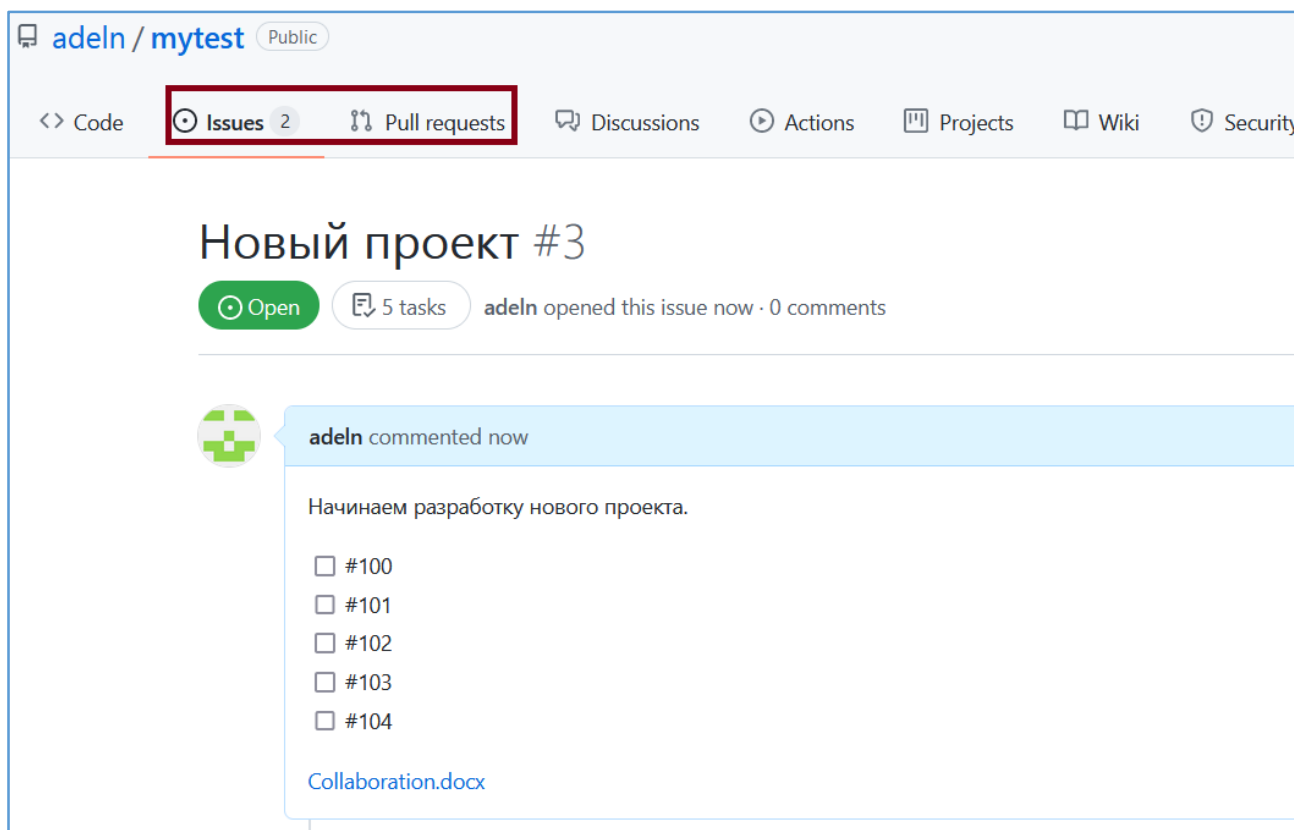
Имя	Дата изменения	Тип	Размер
.git	10.11.2021 20:41	Папка с файлами	
Collaboration.docx	10.11.2021 20:31	Документ Microso...	13 КБ
README.md	10.11.2021 19:23	Файл "MD"	1 КБ



f. Назначение проблем и задач другим пользователям

Можно назначить до 10 человек для решения каждой проблемы, включая вас самих, и всех, у кого есть разрешения на запись в репозиторий.

Создаем новый проект, добавляем его описание и список задач, каждой из которых можно назначить исполнителя.



Отображение назначенной проблемы у соавтора:

ivanovmixa / testPublic

Unwatch

<> Code

Issues 1

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Issue 1 #1

Open

ivanovmixa opened this issue 10 hours ago · 0 comments

ivanovmixa commented 10 hours ago

Owner😊⋮

Называем проект!
Может "С
[Collaboration.docx](#)
овместная работа?"

ivanovmixa added the **question** label 10 hours ago

Write

Preview

H

B

I

≡

<>

🔗

☰

☰

☑

@

📎

↶

Assignees

No one—assign yourself

Labels

question

Projects

None yet

Milestone

No milestone

35