



> РАБОЧЕЕ ОКРУЖЕНИЕ

> Об инструментах

Прежде всего давайте познакомимся с рабочим окружением и инструментами, которые мы будем использовать для работы с данными:

- ClickHouse — колоночная база данных для хранения пользовательских событий;
- Redash — инструмент для написания SQL-запросов и базовой визуализации;
- Superset — полноценная BI-система, о ней будет в следующем уроке;
- JupyterLab — среда для написания кода на Python;
- GitLab — репозиторий для хранения кода.

Теперь давайте настроим все инструменты и посмотрим, как они связаны друг с другом.

> ClickHouse и Redash

Начнём с ClickHouse и Redash. Перейдите [по ссылке](#) и войдите в Redash. Давайте посмотрим, к каким таблицам в ClickHouse у нас есть доступ. Выберите нужную схему данных и напишите первый простой запрос, чтобы посмотреть, как сохраняются события просмотров постов.

```
▼  
1 SELECT  
2     *  
3 FROM simulator.feed_actions
```

```
4 WHERE toDate(time) = today() AND action = 'view'
5 ORDER BY time DESC
6 LIMIT 10
```

Давайте посмотрим на последние 10 событий, которые наши пользователи совершили, просматривая посты.

Каждый раз, когда пользователь просматривает или лайкает пост, соответствующее событие сохраняется в таблицу. Поэтому она постоянно пополняется новыми данными об активности пользователей, которые и днём и ночью пользуются нашим приложением.

Вторая таблица, к которой у нас есть доступ, — это данные об отправке сообщений. Изучите эту таблицу самостоятельно с помощью запросов.

При помощи Redash вы можете не только удобно смотреть результаты выполнения запросов, но и визуализировать полученные данные. Давайте посмотрим, как распределилась активность пользователей, просматривавших посты в течение вчерашнего дня.

Чтобы получить количество просмотров по 15-минутным интервалам за вчерашний день, выполним следующий запрос:

```
1 SELECT
2     toStartOfFifteenMinutes(time) as t,
3     count(user_id)
4 FROM simulator.feed_actions
5 WHERE toDate(time) = yesterday() AND action = 'view'
6 GROUP BY t
```

Перейдём в настройки графиков и визуализируем результат. Отложим время по оси ***X*** и число событий по оси ***Y***. Легко заметить суточный паттерн использования приложения. Пик активности приходится на вечер, при этом ночью и утром приложением пользуются не так часто.

Попробуйте самостоятельно поискать какой-нибудь паттерн использования приложения в течение недели.

> JupyterLab

[JupyterLab](#) — среда для написания и выполнения кода, родственная классическому Jupyter Notebook, но с рядом дополнительных возможностей. В

рамках курса мы будем пользоваться его разновидностью под названием JupyterHub — эта версия практически не отличается от обычного JupyterLab, но она адаптирована для использования на сервере. Вы можете найти его [здесь](#), а также в разделе “Инструменты”.

При входе вы увидите список окружений. Выберите то, которое соответствует Симулятору Аналитика (SimDA), и нажмите на Start.

Server Options

<input type="radio"/>	StartML Окружение для курса StartML
<input type="radio"/>	Data Engineer Окружение для курса Инженер Данных
<input type="radio"/>	HardML A/B tests Окружение для модуля 'A/B тестирование' курса HardML
<input type="radio"/>	HardML Pricing Окружение для модуля 'Динамическое ценообразование' курса HardML
<input type="radio"/>	HardML Ranking Окружение для модуля 'Ранжирование и матчинг' курса HardML
<input type="radio"/>	HardML Uplift-Modelling Окружение для модуля 'Uplift-Modelling' курса HardML
<input type="radio"/>	SimAB Окружения для курса Симулятор A/B Тестов
<input type="radio"/>	SimDA Окружение для курса Симулятор Аналитика
<input type="radio"/>	SimML Окружение для курса Симулятор ML
<input type="radio"/>	HardDA Окружение для курса HardDA

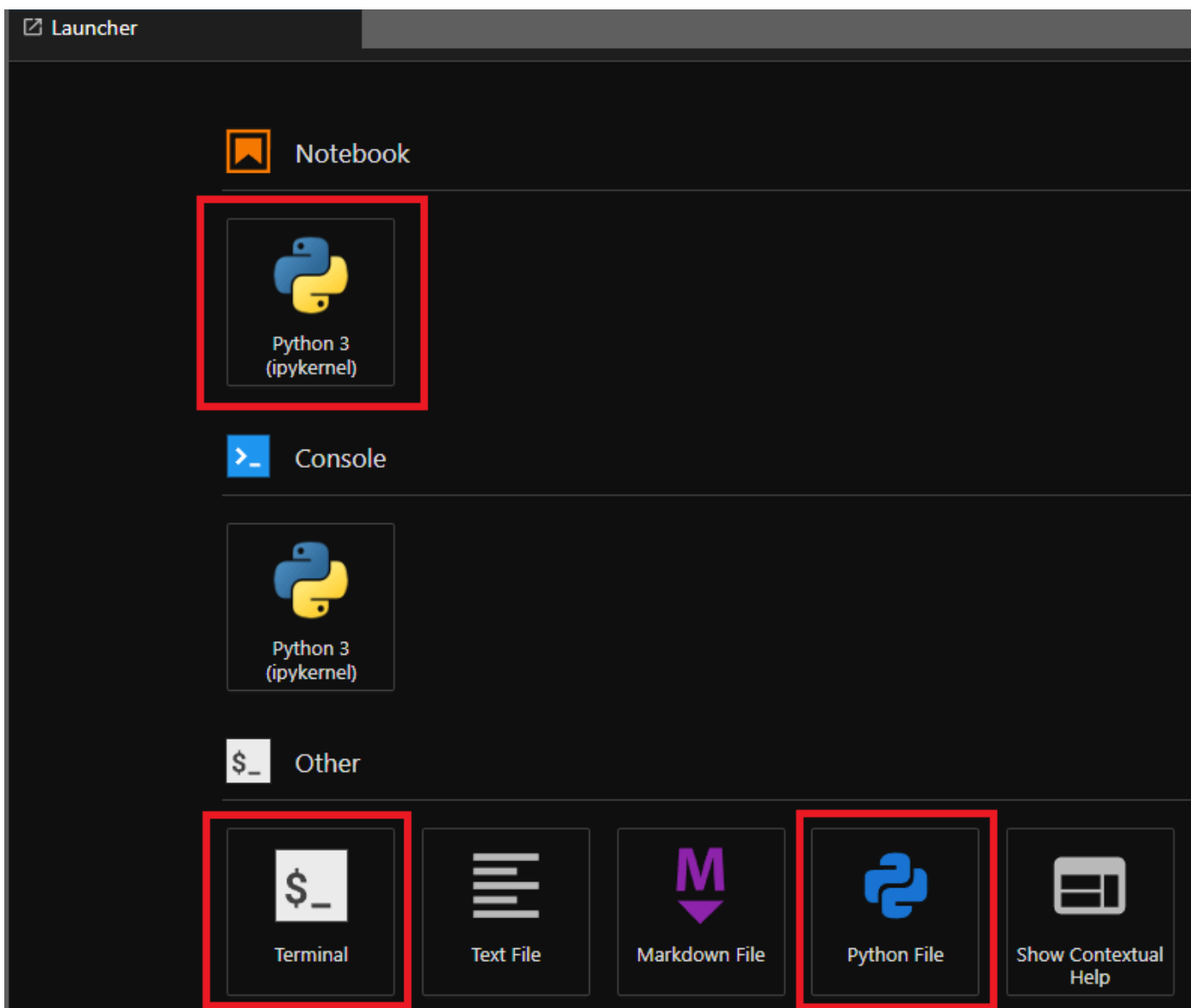
Start

Иногда при длительной работе, а также после выхода из JupyterHub вас может выкинуть из вашего окружения. Это может проявиться в том, что некоторые библиотеки перестанут импортироваться. В таком случае вы можете остановить сервер и войти в него заново следующей последовательностью: File

→ Hub Control Panel → Stop My Server → Start Server, после чего снова выбрать нужное окружение.

Создаваемые файлы

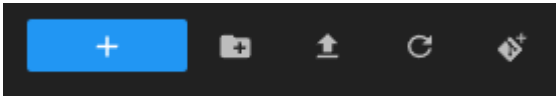
В правой половине интерфейса вы увидите Launcher — страницу, на которой мы сможем создать новый файл определённого типа. Три наиболее полезных типа файлов приведены на скриншоте.



- Python 3 — файл формата `.ipynb`, он же ноутбук. В основном код вы будете писать там.
- Python File — файл формата `.py`. Фактически обычный текстовый файл, в котором можно писать код и который будет восприниматься как код на Python. Этот тип файла будет вам полезен в уроках, связанных с использованием Airflow.

- Terminal — командная строка. Она будет полезна для работы с Git (о нём в следующих шагах).

Также обратите внимание на верхнюю часть левой половины интерфейса:



Слева направо:

- Создать новый Launcher (большая синяя кнопка)
- Создать папку
- Загрузить новый файл
- Обновить отображающиеся файлы
- Клонировать репозиторий (связано с Git, чаще всего не работает)

Установленные библиотеки

Список установленных библиотек и их версий можно посмотреть в терминале с помощью команды `pip list`:

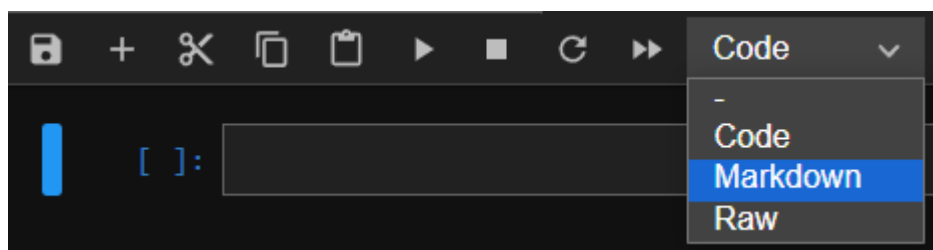
```
Terminal 1
karpov@jupyter-manaenkov-2ea:~$ pip list
Package                               Version
-----
absl-py                               1.4.0
alembic                               1.7.6
anyio                                  3.5.0
APScheduler                           3.6.3
argon2-cffi                           21.3.0
argon2-cffi-bindings                  21.2.0
arviz                                  0.15.1
asttokens                             2.0.5
astunparse                            1.6.3
async-generator                        1.10
attrs                                  21.4.0
Babel                                  2.9.1
```

Вы также можете устанавливать свои библиотеки, но **делать это не рекомендуется** — есть вероятность поломать зависимости и сделать текущие библиотеки неработоспособными. Будьте аккуратны с установкой чего-то нового. Всё самое нужное уже стоит на сервере.

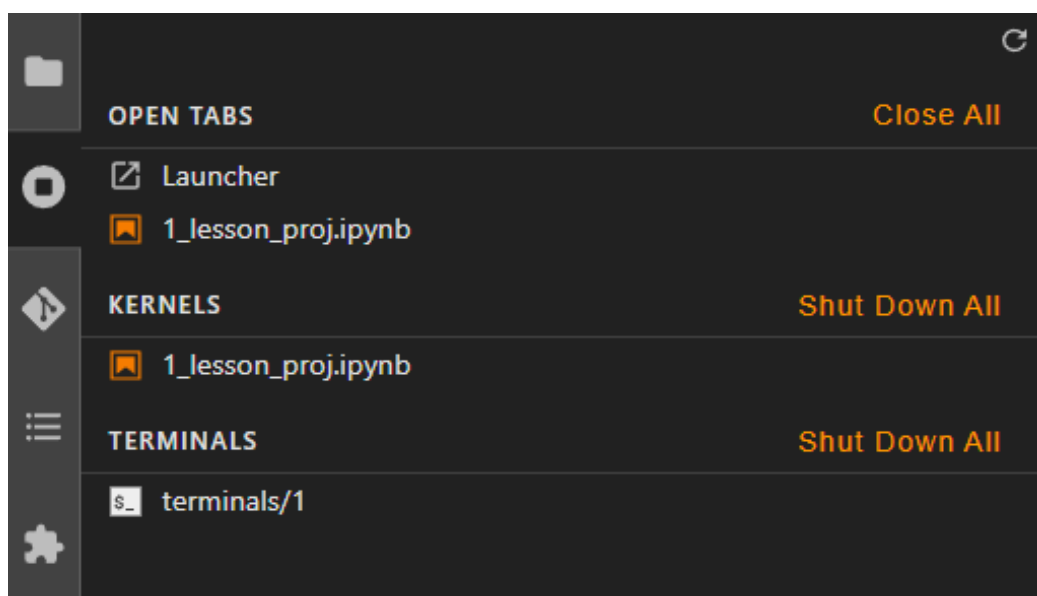
Другие полезные вещи

Основную информацию вы можете найти в документации JupyterLab. Здесь мы расскажем вам о паре вещей, которые вам могут быть полезны в первую очередь.

- Settings позволит вам настроить внешнее отображение и некоторые другие интересные вещи. В частности, там можно поменять тему с белой на чёрную (Theme → Dark) и включить автоматическое закрытие скобок (Auto Close Brackets)
- В разделе Kernel вы можете остановить текущую выполняющуюся ячейку (Interrupt Kernel), а также перезапустить ноутбук целиком (Restart Kernel с разными опциями)
- Если выделить ячейку внутри ноутбука и выбрать в качестве её типа Markdown, то вы можете писать в ней текст и форматировать его. Основной синтаксис форматирования Markdown можно подглядеть [тут](#).



- Если на левой панели выбрать значок в виде черного квадрата внутри белого кружка, то вместо списка файлов и папок вы увидите текущие открытые вкладки, а также активные ноутбуки и терминалы. С этой страницы любой из них можно остановить (Shut Down), чтобы освободить ресурсы.



- Вы можете писать запросы к SQL-таблицам прямо из JupyterHub! Для этого вам понадобится пакет [pandahouse](#) (установлен на сервере).

```








1 import pandas as pd
2 import pandahouse as ph
3
4 #параметры соединения - нужны, чтобы подключиться к нужной схеме данных
5 connection = {'host': 'https://clickhouse.lab.karpov.courses',
6 'database': 'simulator',
7 'user': 'student',
8 'password': 'dpo_python_2020'
9 }
10
11 #текст запроса
12 query = '''
13 select post_id,
14 countIf(action = 'view') as views,
15 countIf(action = 'like') as likes,
16 uniq(user_id) as uniq_users
17 from {db}.feed_actions
18 where toDate(time) = yesterday()
19 group by post_id
20 order by views desc
21 limit 10
22 '''
23
24 # эта функция выполнит запрос и запишет его результат в pandas DataFrame
25 df = ph.read_clickhouse(query, connection=connection)

```

Помимо `read_clickhouse()`, вам могут быть полезны команды `execute()` (выполняет произвольный запрос, полезна для создания пустых таблиц) и `to_clickhouse()` (записывает датафрейм в SQL-таблицу, присоединяя её снизу).

> Git. Полезные команды терминала

Знакома ли вам такая ситуация?

 мой отчёт	docx
 мой отчёт (доработанный)	docx
 мой отчёт (доработанный) (2)	docx
 мой отчёт (доработанный) (2) FINAL	docx
 мой отчёт (доработанный) (2) MEGAFINAL	docx
 мой отчёт (доработанный) (2) ABSOLUTELY FINAL NO CHANGES ACCEPTED	docx
 я пытался уйти от любви	docx

Вечные доработки файлов и создание всё новых и новых версий того же файла — ситуация, которая очень быстро выходит из-под контроля, особенно в командной работе. Разработчики давно обратили внимание на это и в качестве решения проблемы придумали [системы контроля версий](#). Это специальные

программы, которые позволяют отслеживать и сохранять всю историю изменений с возможностью восстановления предыдущих вариантов работы. Наиболее популярной подобной программой является **Git**.

Немного полезных ресурсов:

- [Документация](#)
- [Официальный учебник](#)
- [Забавный гайд](#) по стандартным проблемам и как их решить
- [Визуальный тренажёр](#)

Мы будем пользоваться Git с помощью терминала. Основные команды Git мы рассмотрим в следующих шагах — здесь же мы рассмотрим базовые команды терминала, которые помогут вам в дальнейшем.

ls

Сокращение от *list* — эта команда перечисляет все объекты, которые есть в текущей рабочей директории (грубо говоря, папке, в которой вы находитесь).

```
karpov@jupyter-manaenkov-2ea:~$ ls
1_lesson_proj.ipynb      miniproject_lesson_6-Copy1.ipynb
1_opt_pandas.ipynb      nyc.csv.zip
2_lesson_project.ipynb  'olist_analysis (2).ipynb'
2_resaping_data.ipynb   olist_customers_dataset.csv
3_lesson_project.ipynb  olist_order_items_dataset.csv
3_resample_time.ipynb   olist_orders_dataset.csv
4_style.ipynb           'probability theory.ipynb'
```

cd

Сокращение от *change directory* — эта команда позволяет менять текущую рабочую директорию. Используется как `cd путь_к_папке`. Пути являются относительными, то есть отсчитываются относительно той директории, где вы находитесь в данный момент.

В примере ниже мы переходим в уже существующую папку под названием Simulator. Обратите внимание, как изменилась отображающаяся директория:

```
karpov@jupyter-manaenkov-2ea:~$ cd Simulator
karpov@jupyter-manaenkov-2ea:~/Simulator$ █
```


Чтобы выйти из текущей директории, вам пригодится команда `cd ..` :

```
karpov@jupyter-manaenkov-2ea:~/Simulator$ cd ..  
karpov@jupyter-manaenkov-2ea:~$
```

mkdir

Сокращение от *make directory* — эта команда позволяет создать новую рабочую директорию. Аналогична кнопке New Folder в JupyterLab. Используется как `mkdir имя_папки` .

Ниже мы пробуем войти в несуществующую папку Test, на что терминал выдаёт ошибку об отсутствии такой папки. После её создания с помощью `mkdir` мы можем без проблем войти в неё.

```
karpov@jupyter-manaenkov-2ea:~$ cd Test  
bash: cd: Test: No such file or directory  
karpov@jupyter-manaenkov-2ea:~$ mkdir Test  
karpov@jupyter-manaenkov-2ea:~$ cd Test  
karpov@jupyter-manaenkov-2ea:~/Test$
```

rm

Сокращение от *remove* — эта команда позволяет удалить файл. Используется как `rm имя_файла` . Если хотите удалить папку вместе со всем её содержимым, то нужно указать дополнительный модификатор `-r` — тогда команда будет выглядеть как `rm -r название_папки` .

Ниже мы создали внутри папки Test два объекта: файл `some file.ipynb` и папку `Some folder` . Обратите внимание, что `rm` без проблем удаляет файл, но выдаёт ошибку при удалении папки. `rm -r` удаляет её без проблем.

```
karpov@jupyter-manaenkov-2ea:~/Test$ ls
'some file.ipynb'  'Some folder'
karpov@jupyter-manaenkov-2ea:~/Test$ rm 'some file.ipynb'
karpov@jupyter-manaenkov-2ea:~/Test$ ls
'Some folder'
karpov@jupyter-manaenkov-2ea:~/Test$ rm 'Some folder'
rm: cannot remove 'Some folder': Is a directory
karpov@jupyter-manaenkov-2ea:~/Test$ rm -r 'Some folder'
karpov@jupyter-manaenkov-2ea:~/Test$ ls
karpov@jupyter-manaenkov-2ea:~/Test$
```

Пара дополнительных замечаний:

- Обратите внимание, что названия объектов выше обрамлены в кавычки. Это связано с наличием пробела в названиях; если бы мы назвали свои объекты `some_file.ipynb` и `Some_folder`, то их не пришлось бы ставить. В целом избегать пробелов в названиях — хорошая практика, особенно в контексте Git.
- `rm` удаляет объекты насовсем. **Если вы что-то удалили, то вернуть это нельзя.**
- Многие команды терминала имеют дополнительные опции, меняющие результат выполнения этой команды — `rm -r` является одним из таких вариантов. Можете поискать в интернете документацию всех описанных здесь команд и посмотреть, какие опции есть у каждой.

> GitLab. Создание ключей. git clone

Git можно использовать индивидуально — установить его к себе на компьютер, создать папку и отслеживать изменения файлов в ней. Специально выделенная папка, в которой Git документирует изменения, называется *репозиторием*.

Однако реальная сила Git заключается в совместной работе, когда сразу у нескольких человек есть доступ к некоторой папке (часто находящейся на облачном сервисе), в которой каждый выполняет свою работу. Такая папка называется *удалённым репозиторием*.

Специально для Git были созданы особые платформы, предоставляющие удобный интерфейс для работы с удалёнными репозиториями. Наиболее известной среди них является [GitHub](#). Однако в нашем случае мы будем пользоваться другой платформой, а именно **GitLab**. Как и JupyterHub, GitLab

установлен на наших серверах и предоставляет доступ к удалённым репозиториям Karrov.Courses. Найти его можно по [этой ссылке](#), а также в разделе “Инструменты”.

Однако чтобы пользоваться GitLab, вам нужно подружить его со своим JupyterHub. Для этого вам понадобится создать ключи.

Ключи

[SSH-ключи](#) — это своего рода пароли. Кодовые слова, которыми обмениваются GitLab и Git на вашем JupyterHub, чтобы опознать друг друга. Сгенерировать их можно с помощью команды `ssh-keygen -t ed25519`.

Распишем по шагам, что вам нужно сделать:

- Выполняете команду `ssh-keygen -t ed25519`
- Первое, что вам предложит терминал — указать место, куда сохранить ключ. **Ничего не указывайте, просто нажмите Enter.**
- Затем он попросит создать пароль — придумываете и вводите его. Обратите внимание, что написание паролей не выводится на экран из соображений безопасности; не пугайтесь того, что пароль будто бы не пишется.
- Потом он попросит подтвердить пароль — вбиваете его повторно. **Обязательно запомните или запишите его:** если вы забудете пароль, то вам придётся удалить ключи и создать их заново.
- Готово, ваш ключ создан и сохранён в скрытой папке `.ssh` !

```

karpov@jupyter-manaenkov-2ea:~$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/karpov/.ssh/id_ed25519):
Created directory '/home/karpov/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/karpov/.ssh/id_ed25519
Your public key has been saved in /home/karpov/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:TsCUGVsY781PLBh61YeUfvav53qh4XUKanVvxH0ZMJk karpov@jupyter-manaenkov-2ea
The key's randomart image is:
+--[ED25519 256]--+
|      +* .   ..o  |
|    o++   o.E    |
|      + o ..o +   |
|      + *   ...o. |
|      . S + oo o+ |
|      +   +o..+B  |
|      .   ooo++=  |
|      o   o. *    |
|      .   oB.    |
+-----[SHA256]-----+

```

Любой SSH-ключ состоит из двух частей: приватной и публичной. Публичная часть нам и понадобится для связи с GitLab. Чтобы её получить, вам нужно сделать следующее:

- Перейдите в папку `.ssh`. Внутри неё будут два файла — `id_ed25519` и `id_ed25519.pub`. Нам нужен второй.
- Выполните команду `cat id_ed25519.pub` — она напечатает содержимое публичного ключа. Скопируйте всю последовательность, которую вам напишет терминал.

```

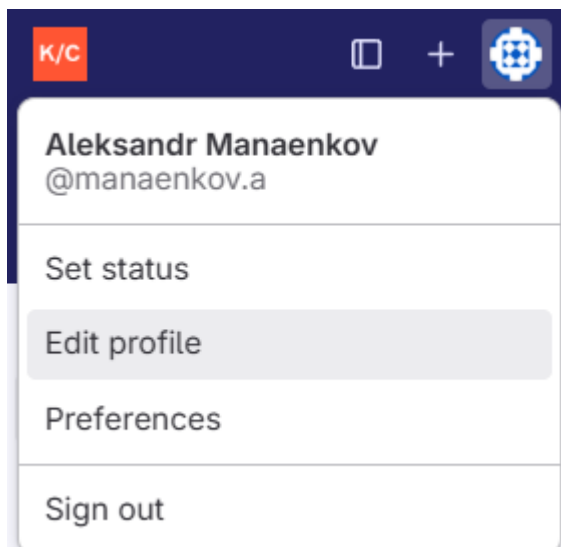
karpov@jupyter-manaenkov-2ea:~$ cd .ssh
karpov@jupyter-manaenkov-2ea:~/.ssh$ ls
id_ed25519  id_ed25519.pub
karpov@jupyter-manaenkov-2ea:~/.ssh$ cat id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIG3EkvhJT/ZHxn3nyJJCoY8W1I1gPTW9yJZ85zE0Cz+t
karpov@jupyter-manaenkov-2ea

```

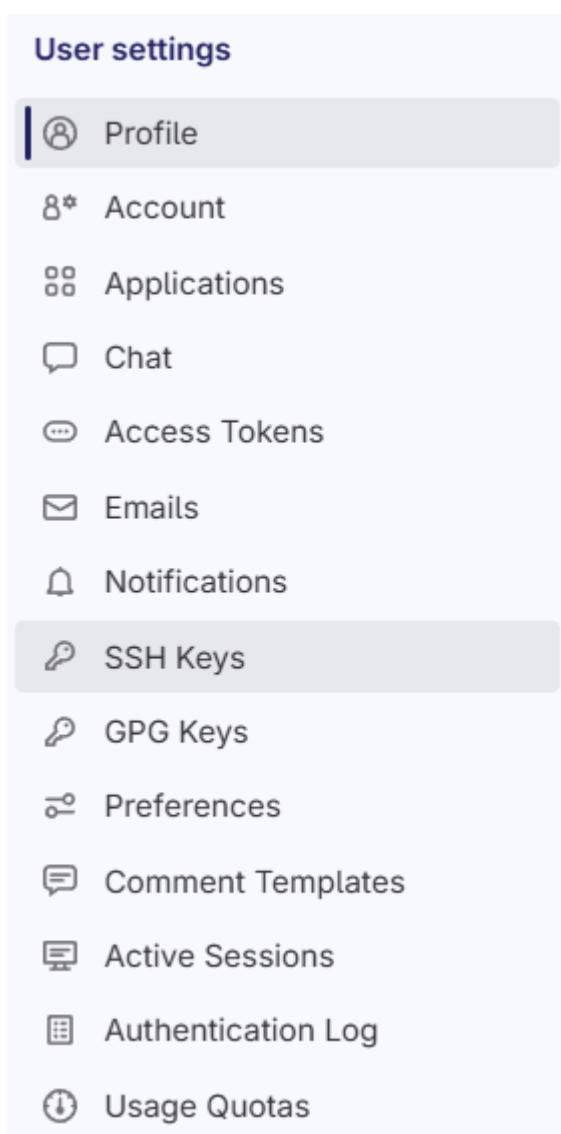
ВНИМАНИЕ: никогда и никому не показывайте приватную часть ключа.

Публичную показывать безопасно, а вот приватную не должен видеть никто.

После того, как вы получили публичный ключ, переходите в GitLab. В своём профиле (левый верхний угол) выберите опцию *Edit profile*.








Слева выберите опцию *SSH Keys*:



Нажимаете на *Add new key*:

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab. SSH fingerprints verify that the client is connecting to the correct host. Check the [current instance configuration](#).

Your SSH keys 4							Add new key
Title	Key	Usage type	Created	Last used	Expires	Actions	
karpov@jupyter-manaenkov-2ea		Authentication & Signing	8 months ago	2 months ago	Never	Revoke	
desktop key		Authentication & Signing	3 years ago	2 years ago	Never	Revoke	
my laptop		Authentication & Signing	3 years ago	Never	Never	Revoke	
jupyter-manaenkov.a@lab.karpov.courses		Authentication & Signing	4 years ago	2 months ago	Never	Revoke	

В окошечке *Key* указываете весь публичный ключ, после чего нажимаете на кнопку *Add key*. Теперь вы можете получить доступ к удалённым репозиториям и пользоваться GitLab!

Add an SSH key

Add an SSH key for secure access to GitLab. [Learn more](#).

Key

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBUQ2/AwZIEBDWLUBU2d/9nqFHqYgywl7xU1Wu7WvTPi karpov@jupyter-manaenkov-2ea

Begins with 'ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

Title

karpov@jupyter-manaenkov-2ea

Key titles are publicly visible.

Usage type

Authentication & Signing

Expiration date

2026-05-19

Optional but recommended. If set, key becomes invalid on the specified date.

Add key Cancel

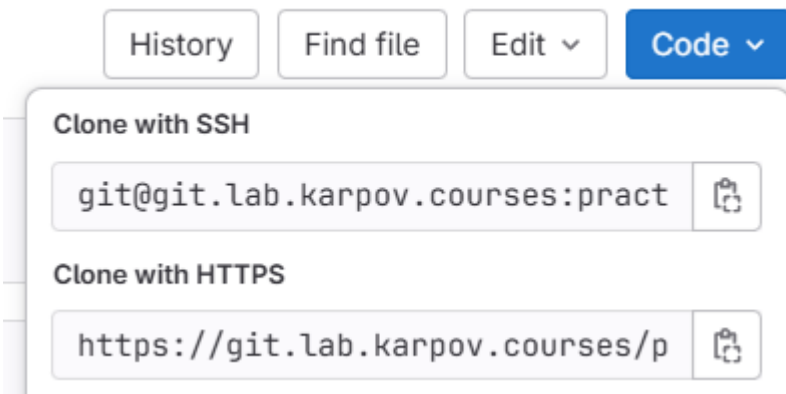
Клонирование репозитория

На [главной странице](#) вы можете получить список всех доступных вам удалённых репозиторий. Для начала нас интересует репозиторий под названием [analyst_simulator](#).

Первое, что нам надо сделать — это **клонировать** этот репозиторий. По сути это создание копии папки вместе с её содержимым, которая будет находиться у вас в JupyterHub и с которой можно взаимодействовать как с обычной папкой.

Чтобы это сделать, вам нужно выполнить следующее:

- Нажимаете на синюю кнопку *Code*. Копируете ссылку, указанную в разделе Clone with SSH — в данном случае это `git@git.lab.karpov.courses:practice-da/analyst_simulator.git`



- Вернитесь в корневую папку. В ней выполните команду `git clone ssh-ссылка` — это запустит процесс клонирования. В данном случае это команда `git clone git@git.lab.karpov.courses:practice-da/analyst_simulator.git`
- Терминал спросит вас, уверены ли вы, что хотите клонировать — пишете `yes` и нажимаете Enter. **Если терминал пишет вам “Please tell me who you are” — загляните в следующий шаг, там есть решение этой проблемы.**
- Вводите пароль, который вы создали для своего ключа.
- Готово, теперь вы можете увидеть папку `analyst_simulator` у себя в JupyterHub!

```
karpov@jupyter-manaenkov-2ea:~$ git clone git@git.lab.karpov.courses:practice-da/analyst_simulator.git
Cloning into 'analyst_simulator'...
The authenticity of host 'git.lab.karpov.courses (116.203.248.181)' can't be established.
ECDSA key fingerprint is SHA256:zoixDAa76yae+AZIH2iRbkDgq26WP0fFI48AvVx6Two.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'git.lab.karpov.courses,116.203.248.181' (ECDSA) to the list of known hosts.
Enter passphrase for key '/home/karpov/.ssh/id_ed25519':
remote: Enumerating objects: 6940, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (29/29), done.
remote: Total 6940 (delta 5), reused 0 (delta 0), pack-reused 6911
Receiving objects: 100% (6940/6940), 108.61 MiB | 19.45 MiB/s, done.
Resolving deltas: 100% (2657/2657), done.
karpov@jupyter-manaenkov-2ea:~$
```


> Частые проблемы

1. Первая распространённая проблема, пугающая новичков — `git clone` или другая команда не выполняется, а вместо этого пишется вот такое сообщение:

```
1 *** Please tell me who you are.
2
3 Run
4
5 git config --global user.email "you@example.com"
6 git config --global user.name "Your Name"
7
8 to set your account's default identity.
9 Omit --global to set the identity only in this repository.
```

Это происходит в том случае, если вы впервые используете Git на этой машине — он хочет с вами познакомиться. Выполните дословно обе команды, которые он вам пишет, но вместо `you@example.com` укажите свою почту, а вместо `Your Name` — своё имя латиницей. После этого повторите изначальную команду, она должна заработать. Сделать это нужно только один раз.

1. Иногда при попытке выполнить любую команду возникает вот такая ошибка:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@          WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0660 for '/home/karpov/.ssh/id_ed25519' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "/home/karpov/.ssh/id_ed25519": bad permissions
git@git.lab.karpov.courses: Permission denied (publickey).
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

Эта ошибка связана с излишне широкими правами доступа вашего ключа. Чтобы её решить, выполните команду `chmod 400 ~/.ssh/id_ed25519`, после чего повторите вашу изначальную команду. Эта ошибка может возникнуть больше одного раза — во всех случаях выполняете команду выше.

1. Вы можете натолкнуться на ошибку `fatal: not a git repository`. Это означает, что вы забыли зайти внутрь репозитория — сделайте это с помощью команды `cd`.

2. Изредка команды `git clone` и `git push` могут отказаться работать без объективных причин. В таком случае помогает удаление ключей из папки `.ssh` и с GitLab — после удаления создайте ключи заново по схеме из прошлого шага, и всё должно заработать.

> git branch и git checkout

Одна из распространённых рабочих ситуаций — вы пишете отчёт, но у разных коллег есть своё мнение насчёт того, что именно должно быть в этом отчёте. Кому-то хочется отчёт с графиками, кому-то хочется отчёт с табличками, а что в итоге нужно генеральному директору — вообще загадка. В результате вы держите под рукой две или даже больше альтернативных версий одного и того же отчёта и ждёте, какая из них в итоге будет основной.

 [.]	
 отчёт с табличками	docx
 отчёт с графиками	docx

Похожая идея реализуется в Git через концепцию *веток* — альтернативных версий одного и того же репозитория. В любом репозитории всегда есть как минимум одна ветка — **main** (в старых версиях Git она называлась **master**). Это основное состояние репозитория: в ветке `main` всегда находится стабильная версия приложения, работоспособная библиотека, финальный вариант отчёта и так далее.

Другие ветки часто содержат в себе новый код, которого в основной ветке ещё не существует — часто там тестируется новый функционал. Здесь может быть полезна аналогия альтернативных вселенных: если ветка `main` является основным таймлайном, то другие ветки являются версиями той же вселенной, где что-то когда-то пошло совсем по другому пути.

Именно через ветки чаще всего и реализуется командная работа: программист создаёт ветку и пишет в ней новый код. Если результат полезен для проекта, то эта ветка объединяется с основной; если в коде есть ошибки или он не нужен — код дописывается до работоспособного состояния или так и остаётся в отдельной ветке. Так обеспечивается стабильная работа над проектом и пониженный риск что-то сломать — а если что-то и сломается, то всегда можно отследить, в какой момент это произошло и кто виноват.

Все работы, в которых от вас потребуется Git, потребуют также создание собственных веток. Давайте посмотрим, как это делать!

Как пользоваться ветками?

Первый шаг — нам нужно войти в тот репозиторий, где мы планируем создавать ветки. Иначе команды не будут работать (лучший случай) или непредсказуемым способом выстрелят в ногу (худший случай). Войдём в клонированный нами репозиторий `analyst_simulator`:

```
karpov@jupyter-manaenkov-2ea:~$ cd analyst_simulator
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$
```

Чтобы посмотреть, какие ветки есть в клонированном репозитории, нам понадобится команда `git branch`. В данный момент в клонированном репозитории есть только ветка `main`.

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git branch
* main
```

Чтобы создать новую ветку, нам нужно выполнить команду `git branch` `название_ветки`. Создадим ветку с названием `my_precious`; если мы после этого снова выполним команду `git branch` без аргументов, то мы увидим две ветки:

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git branch my_precious
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git branch
* main
  my_precious
```

Вам лучше создать ветку с каким-нибудь другим, уникальным названием. Это нужно для того, чтобы в GitLab потом не возникало конфликта между двумя ветками с одинаковым названием.

Чтобы перейти в другую ветку, нам нужно выполнить команду `git checkout` `название_ветки`. Обратите внимание, что теперь звёздочка стоит рядом с названием `my_precious`, а само название теперь выделено зелёным цветом:

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git checkout my_precious
Switched to branch 'my_precious'
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git branch
  main
* my_precious
```

Всегда проверяйте, что вы находитесь в своей ветке и выполняете все последующие действия именно там!

> git status и git add

Итак, мы клонировали репозиторий и создали ветку. В этом репозитории у нас четыре объекта: два скрипта на питоне, папка `read_db` и маркдаун-файл `README.md`.

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ ls
example_alert.py  read_db  README.md  test_report.py
```

Новая команда, которая будет нам полезна — это `git status`. Она показывает все изменения, которые произошли в репозитории с момента его клонирования. На данный момент мы ничего не сделали, поэтому эта команда ничего интересного не покажет — только текущую ветку и то, что делать особо нечего.

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
nothing to commit, working tree clean
```

Давайте внесём два изменения:

1. Добавим лишнюю строчку кода в файл `test_report.py`
2. Создадим новый файл `my_big_work.ipynb`

Теперь `git status` показывает, что что-то изменилось:

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
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:   test_report.py

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

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

Вы можете видеть, что файлы разделились на две категории:

- *Untracked files* — файлы, которых раньше не было в этом репозитории. В данном случае это свежесозданный файл `my_big_work.ipynb`, а также

временная папка `.ipynb_checkpoints/`

- *Modified* — изменённые файлы. Эти файлы не новые; Git знает, что они уже были в этом репозитории, но в них что-то изменилось. В данном случае это `test_report.py`.

Пока Git только заметил эти изменения — теперь нам нужно показать, какие из этих изменений необходимо сохранить в истории. Чтобы это сделать, нам нужно выполнить команду `git add название_файла`.

Допустим, мы хотим оставить в истории только новый файл `my_big_work.ipynb`. От временных файлов `.ipynb_checkpoints/` толку мало, а `test_report.py` мы вообще изменили по ошибке и не хотим, чтобы Git это запоминал. Поэтому мы делаем только `git add my_big_work.ipynb`:

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git add my_big_work.ipynb
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   my_big_work.ipynb

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:   test_report.py

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

Обратите внимание, что теперь наш файл выделен зелёным цветом, помечен как *new file* и находится в разделе *Changes to be committed*. Теперь этот файл и его содержимое готовы к тому, чтобы остаться в истории!

Рекомендация на будущее: добавляйте через `git add` только те файлы, которые вы действительно хотите сохранить в памяти Git. Если вы не меняли какой-то из файлов лично — не трогайте его. Это будет особенно важно в блоках, связанных с Airflow.

Но допустим, вы случайно добавили какой-то лишний файл и вовремя это заметили. Отменить `git add` довольно просто — выполните команду `git reset название_файла`, и файл вернётся в исходный статус.

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git reset my_big_work.ipynb
Unstaged changes after reset:
M      test_report.py
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
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:   test_report.py

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

> git commit и git log

В прошлый раз мы остановились на том, что добавили наш новый файл `my_big_work.ipynb`. Настало время сохранить его в истории с помощью *коммита*!

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git add my_big_work.ipynb
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   my_big_work.ipynb

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:   test_report.py

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

В этом нам поможет команда `git commit -m "текст сообщения"` — когда вы её выполните, изменения во всех добавленных с помощью `git add` файлов будут зафиксированы Git и помечены тем сообщением, которое вы напишете. Сообщение должно кратко, но ёмко описывать, что именно вы сделали в этом коммите.

Ниже мы создаём коммит с сообщением “Added new file”. Обратите внимание, что наш новый файл исчез из вывода `git status`.


```

karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git commit -m "Added new file"
[my_precious d22f73c] Added new file
 1 file changed, 33 insertions(+)
 create mode 100644 my_big_work.ipynb
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
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:   test_report.py

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

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

```

Созданный коммит будет привязан исключительно к той ветке, на которой вы его выполнили. Мы можете видеть, что наш новый файл отображается в ветке `my_precious`, но абсолютно невидим для ветки `main`.

```

karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git branch
main
* my_precious
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ ls
example_alert.py  my_big_work.ipynb  read_db  README.md  test_report.py
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git checkout main
M       test_report.py
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ ls
example_alert.py  read_db  README.md  test_report.py

```

Вы можете посмотреть на всю историю коммитов в этом репозитории. Для этого вам нужно выполнить команду `git log`.

- В самом верху будет отображаться ваш последний коммит с информацией о ветке, о авторе, времени коммита и его сообщении
- Ниже вы увидите список веток, находящихся в удалённом репозитории
- Если нажать на Enter, то история коммитов начнётся проматываться, пока не дойдёт до самого первого коммита
- Чтобы выйти, нажмите клавишу `q`

```

karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git log
commit fe38426dc1ab7e70772e55591a7cc7e8f45a36c3 (HEAD -> my_precious)
Author: Alexander Manaenkov <manaenkov.a@team.karpov.courses>
Date: Mon Aug 7 16:39:52 2023 +0000

    Added new file

commit 671f9d65f44bf014eb6eb2807079e2b89efb2f11 (origin/yulia_ak.task5, origin/ha
tever, origin/vpiakarski_auto_report, origin/update-code, origin/toloknov_task_3,
origin/tkrupen/ab_test, origin/simulador_51, origin/s.akinfiyeva/lesson5_task1, o
rigin/ruslan_huretski_20.06_r.huretski-8/lesson_5_ab_tests, origin/rbagirov_branc
h, origin/rbagirov, origin/product_metrics, origin/pgolubeva/new_branch, origin/o
gnea/AB_3, origin/ofedorova/les7, origin/narkhipov_branch3, origin/narkhipov_bra
nch2, origin/narkhipov_branch1, origin/my_branch_new, origin/mterentev_DAsim, ori
gin/main, origin/mabutraba/branch_abtest_task1, origin/lesson_6_product_metrics,
origin/lesson_5_AA_test, origin/l-grubin-6/ab-test, origin/l-2dvasileva-2d24, ori
gin/l-2dvasileva-24, origin/kuvandykova/lesson_5_1, origin/kseniya_levinskaya/les
son5_task1, origin/kostarev_ab_test_1, origin/kmaltseva/5_3_aatest, origin/k-begu
nov-18_A/B_tests, origin/jakimtseva12/task, origin/g, origin/ebaburina/new_branch
, origin/bz/test_branch, origin/bot_daily_reports, origin/artiomov/new_branch, or
igin/aperepelova/4modul_aa_test, origin/agusev/new_branch, origin/aburlakov/prob_
branch, origin/abu_traba_ab_test, origin/abtest_abutraba, origin/ab_tests, origin
/Volodin_Alex, origin/MelnikovDV, origin/HEAD, origin/Dmitry_Kozhin_ABtest_task_3
, origin/Dmitry_Kozhin, origin/Chepenko-ABTest, origin/Autoreport, origin/Agile,
origin/AB_test_L, origin/AAA_branch, origin/5.1_Ishberdina, origin/5.1, main)
:

```

Если вы поняли, что поторопились с коммитом, то его также можно отменить. Один из вариантов это сделать — команда `git reset --soft HEAD~1`. Она отменяет последний коммит, но сохраняет все изменения. Другие способы отмены коммита можно увидеть [тут](#).

```

karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git reset --soft HEAD~1
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git status
On branch my_precious
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   my_big_work.ipynb

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:   test_report.py

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

```

Что будет, если не указывать сообщение?

Допустим, вы написали просто `git commit` без указания сообщения. Возможно, вам стало любопытно, что произойдёт. Возможно, вы забыли это сделать. А может, вы решили, что не будете слушаться приказов программы, чьё название переводится с английского как “[мерзавец](#)”. Интересный факт: последний пункт не случайность, а [осознанное решение](#) создателя Git!

Вне зависимости от причины, перед вами появится вот это:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch my_precious
# Changes to be committed:
#   new file:   my_big_work.ipynb
#
# Changes not staged for commit:
#   modified:   test_report.py
#
# Untracked files:
#   .ipynb_checkpoints/
#
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~/analyst_simulator/.git/COMMIT_EDITMSG 14L, 330C      1,0-1      All
```

Что произошло? Если пользователь не указывает сообщение с помощью `-m`, Git открывает текстовый редактор и заставляет пользователя написать это сообщение (иначе коммит отменяется). Текстовый редактор в нашем JupyterHub — это [Vim](#), печально известный благодаря мему “как выйти из Vim?”. Если вы это увидели, то наверняка уже поняли, откуда этот мем взялся.

Как же выйти из Vim? Выполните несколько простых шагов:

1. Нажмите клавишу `i` — это включит режим ввода текста.
2. Введите сообщение коммита.

3. Нажмите клавишу Esc — вы выйдете из режима ввода текста.
4. Введите команду `:wq` и нажмите Enter — это сохранит ваш текст и вы выйдете из Vim.

Результат будет аналогичен `-m "сообщение"`. Если вы больше никогда в жизни не хотите видеть Vim — не забывайте указывать сообщение сразу 😊

> git push

Всё, что вы делали в последних трёх шагах, касалось лишь вашей локальной копии репозитория. Осталось поделиться своими наработками с миром и отправить свою ветку в удалённый репозиторий! В этом нам поможет команда `git push origin название_ветки` — *origin* в данном случае обозначает оригинальный репозиторий, копию которого мы сделали с помощью `git clone` и с которой мы работали всё это время.

После выполнения команды вам опять потребуется ввести пароль — и ваша ветка будет *запущена* в удалённый репозиторий! Остаётся лишь последний шаг.

```
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$ git push origin my_precious
Enter passphrase for key '/home/karpov/.ssh/id_ed25519':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 611 bytes | 611.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote: To create a merge request for my_precious, visit:
remote:   https://git.lab.karpov.courses/practice-da/analyst_simulator/-/merge_requests/new?merge_request%5Bsource_branch%5D=my_precious
remote:
To git.lab.karpov.courses:practice-da/analyst_simulator.git
 * [new branch]      my_precious -> my_precious
karpov@jupyter-manaenkov-2ea:~/analyst_simulator$
```

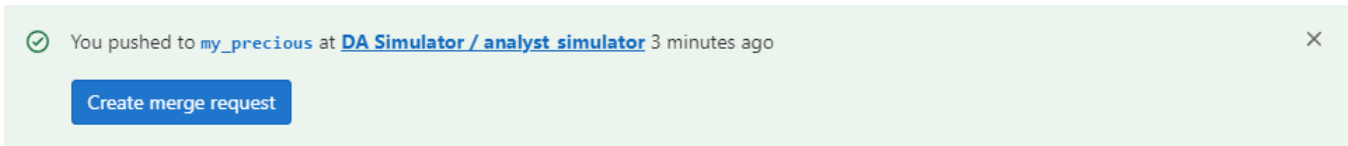
Важно: вы не можете пушить ветку main, если вы не являетесь хозяином репозитория. Поэтому мы и создавали новую ветку.

Merge Request

Последний шаг, который вам остаётся сделать — это создать *запрос на слияние*, он же *Merge Request* (в GitHub это называется *Pull Request*). Для этого вам нужно перейти по ссылке, которую вам даст терминал после `git push` — но

вы также можете просто войти в удалённый репозиторий в GitLab, и вверху будет отображаться вот такое сообщение с синей кнопкой.

DA Simulator > analyst simulator



В любом случае вы попадёте на страницу создания Merge Request. При желании можете указать что-то в описании, но можете просто сразу нажать на кнопку *Create merge request*.

Title

Added new file

Start the title with `Draft:` to prevent a merge request draft from merging before it's ready.

Add [description templates](#) to help your contributors to communicate effectively!

Description

WritePreview

B*I*</>

Describe the goal of the changes and what reviewers should be aware of.

Markdown and quick actions are supported

Attach a file

Assignee

Unassigned

Assign to me

Reviewer

Unassigned

Milestone

Milestone

Labels

Labels

Merge options

☒ Delete source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. ?

Create merge request

Cancel

Готово! Перед вами будет вот такая страница. Если задание потребует от вас ссылку на Merge Request, то просто берёте ссылку из адресной строки и прикрепляете её на LMS.

Added new file

Overview 0 Commits 1 Changes 1

 Request to merge `my_precious` into `main`

Open in Web IDE

Check out branch



 Looks like there's no pipeline here.

GitLab CI/CD can automatically build, test, and deploy your application. It only takes a few minutes to get started, and we can help you create a pipeline configuration file.

Try out GitLab Pipelines



  Approve Approval is optional 



Merge

Ready to be merged automatically. Ask someone with write access to this repository to merge this request

При выполнении заданий, связанных с Airflow, вам также потребуется объединять свою ветку с главной, используя кнопку Merge, но об этом подробнее в соответствующих уроках.