

Как катнуть MR в gitlab



Мы тут разговариваем на языке машин, поэтому все будет через терминал.

Инструкцию, как катить не через терминал мы тоже добавим, но попозже.

Делать изменения через интерфейс gitlab строго запрещается, за это боги дата-инженерии наводят порчу. А если честно – так проще всего что-то поломать, поэтому, пожалуйста, не надо.

Итак. Вы решили внести какое-то изменение из [списка](#) в репозитории DWH. ~~Кто виноват~~ и **что делать?**

Для начала: у вас должна быть заведена задача в Jira, по которой вы делаете свое изменение. Потом уже можно идти в гитлаб, пуллить, пушить и так далее.

- [Настройка интеграции с GIT](#)
- [Клонирование целевого репозитория](#)
- [Работа с репозиторием](#)
 - [Создание своей ветки](#)
 - [Правила оформления MR](#)
 - [Сохранение и отправление изменений на сервер](#)
 - [Создание merge request в интерфейсе гитлаба](#)
 - [Прочие полезные команды](#)

Настройка интеграции с GIT

Для обращения к gitlab по SSH нужно настроить ключи – показать gitlab-у, что ваш личный комп будет туда ходить.

Если вы еще не генерировали SSH-ключи, нужно начать с генерации ключа по шагам 1-2.

Если у вас уже есть публичный SSH-ключ, нужно сразу идти на шаг 3.

Для этого:

1. Заходим в терминал и пишем команду (не забываем заменить адрес электронной почты на свой)

```
ssh-keygen -C "vasya.pupkin@sbermarket.ru"
```

2. Нажимаем три раза Enter: оставляем дефолтное место хранения ключа и не ставим пароль к файлу

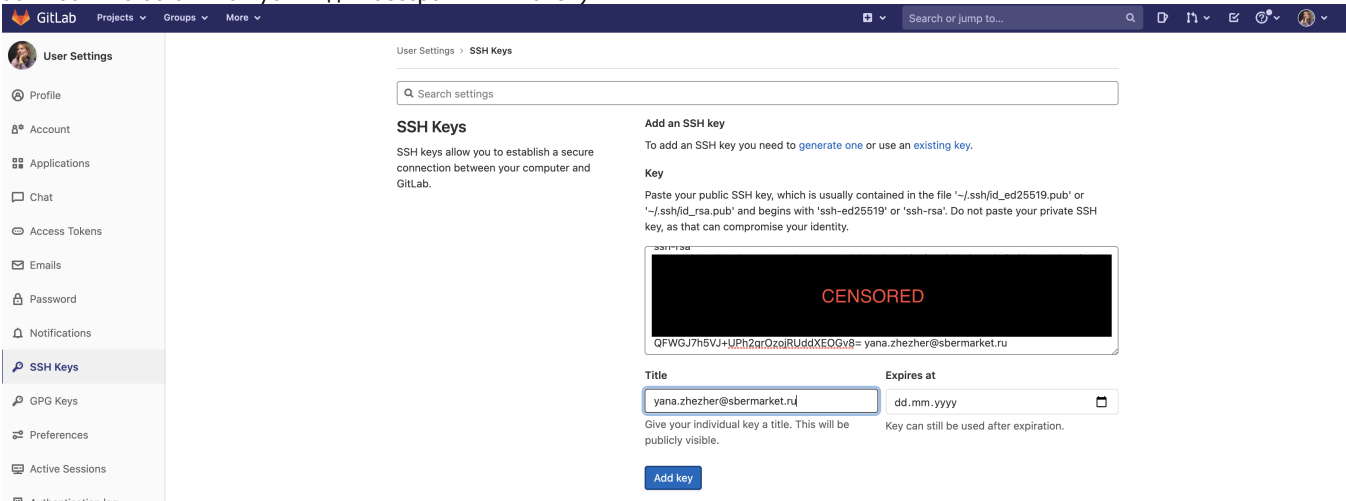
```
(base) yanazhezher@YANAs-MacBook-Pro ~ % ssh-keygen -C "yana.zhezher@sbermarket.ru"
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/yanazhezher/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/yanazhezher/.ssh/id_rsa.
Your public key has been saved in /Users/yanazhezher/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:G09u52c9+F38naczAFnU0mWmo2JhsKF+J/3Cq1X2KXI yana.zhezher@sbermarket.ru
The key's randomart image is:
+---[RSA 3072]---+
|                 .+..+
|                o  .+
|               ..+ o *
|              .+o o =
|             .oS. o o
|            .O+=.... .
|           .O=Eoo + o
|          .++..o + =.*
|         ...o.+..o.O*=
+-----[SHA256]-----+
```

3. Пишем в терминале

```
cat ~/.ssh/id_rsa.pub
```

Что выведет 5-6 строк с крокозябрами, которые заканчиваются вашим адресом электронной почты. Копируем это великолепие и идем в gitlab на страницу <https://gitlab.sbermarket.tech/-/profile/keys>

4. На странице "SSH keys" копируем ключ в окошко "Key", присваиваем ему какое-то название и добавляем ключ (поле "Expires at" я обычно оставляю пустым для бессрочных ключей).



На почту придет письмо о добавлении нового ключа, и теперь можно приступать к удаленной работе с репозиторием.

Клонирование целевого репозитория

1. Заходим в gitlab на страницу с персональными токенами https://gitlab.sbermarket.tech/-/profile/personal_access_tokens. Токен нужно будет получить один раз для того, чтобы подключиться к репозиториям в gitlab удаленно. Здесь пишем название токена, выбираем нужные права (можно просто выбрать все и не париться), оставляем дату пустой, чтобы сделать токен бессрочным – или впишите какую-то дату.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Enter the name of your application, and we'll return a unique personal access token.

Name

gitlab-token

Expires at

YYYY-MM-DD

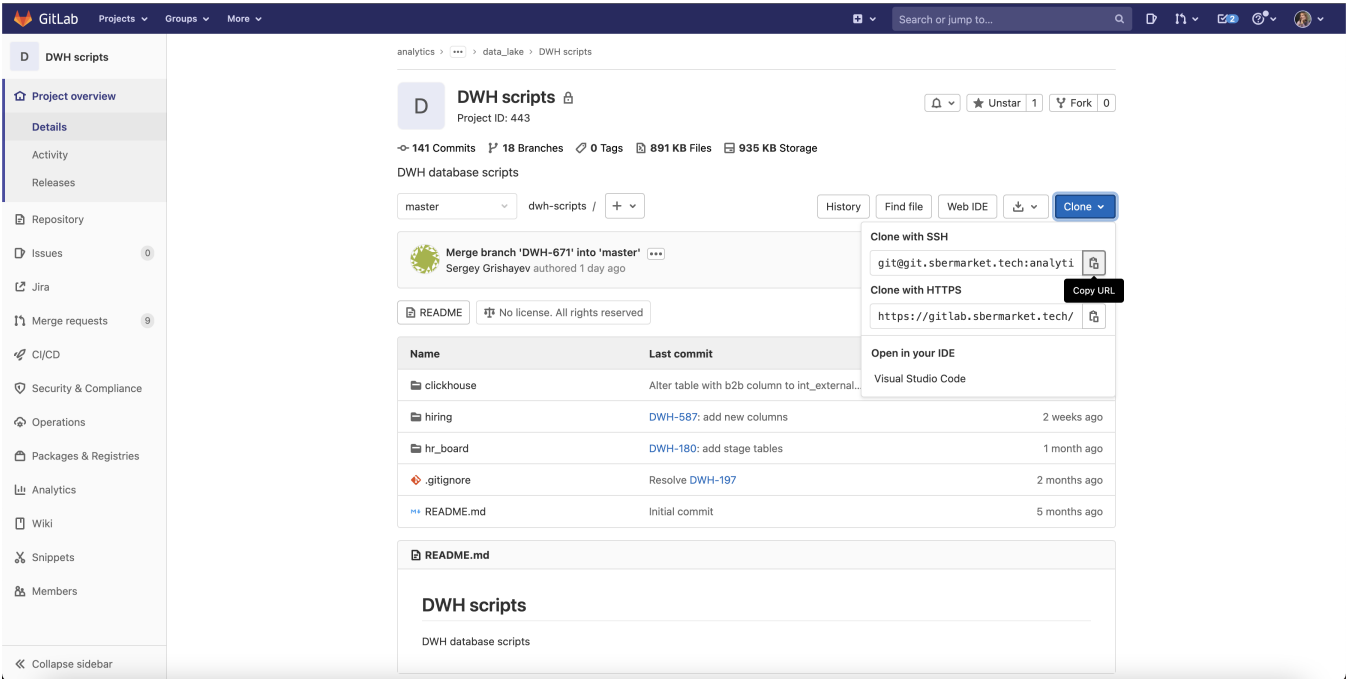
Scopes

- ☒ **api**
Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **read_user**
Grants read-only access to the authenticated user's profile through the User API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.
- ☒ **read_api**
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **read_repository**
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- ☒ **write_repository**
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).
- ☒ **read_registry**
Grants read-only access to container registry images on private projects.
- ☒ **write_registry**
Grants write access to container registry images on private projects.

Create personal access token

Тыкаем на "Create personal access token", копируем появившийся токен куда-то (его не восстановить!).

2. Идем в репозиторий DWH, например [dwh-scripts](#).
3. Нажимаем на кнопку "Clone" и копируем то, что в окошке "Clone with SSH" (либо используем вариант для IDE)



4. Теперь самое страшное: идем в терминал. Выполняем там следующие команды:

```
git init
git clone ,
Username: .
Password: SSH
```

В первый раз появится аутентификация, в котором в качестве логина вводим имя.фамилия, в качестве пароля – ключ SSH. Затем будет работать без пароля.

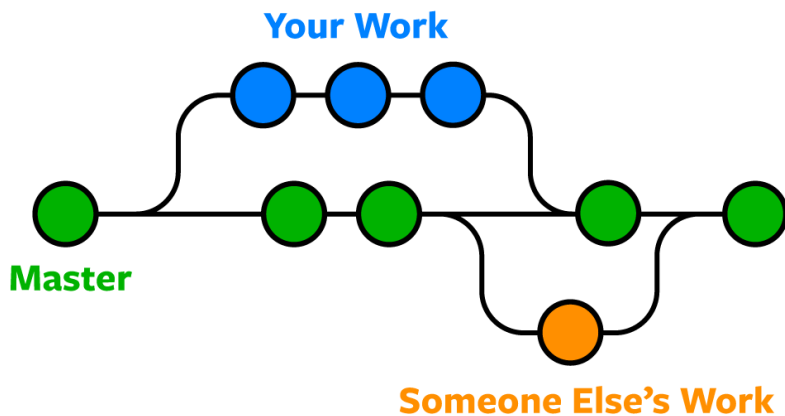
```
((base) yanazhezher@YANAs-MacBook-Pro SBERMARKET % git init
Reinitialized existing Git repository in /Users/yanazhezher/Insync/zhezher.yana@physics.msu.ru/Google Drive/SBERMARKET/.git/
((base) yanazhezher@YANAs-MacBook-Pro SBERMARKET % git clone git@git.sbermarket.tech:analytics/datazone/data_lake/dwh-scripts.git
Cloning into 'dwh-scripts'...
remote: Enumerating objects: 256, done.
remote: Counting objects: 100% (256/256), done.
remote: Compressing objects: 100% (75/75), done.
remote: Total 1580 (delta 214), reused 213 (delta 181), pack-reused 1324
Receiving objects: 100% (1580/1580), 337.43 KiB | 4.50 MiB/s, done.
Resolving deltas: 100% (1127/1127), done.
((base) yanazhezher@YANAs-MacBook-Pro SBERMARKET %
```

Вы скопировали себе на локальную машину папку с названием (в моем случае) dwh-scripts, в которой лежит полный репозиторий.

Работа с репозиторием

Репозиторий (от англ. *repository* — хранилище) — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

В репозитории всегда есть основной код, который сейчас используется на проде. Он всегда называется *master*.



Для внесения изменений в gitlab поддерживается *ветвление*. Вы копируете себе текущую версию master-ветки, и можете параллельно в ней работать, что-то изменять и тестировать. Затем, когда ваш код отлажен, можно обратно его объединить с веткой master – или, говоря современным языком, сделать *merge request*.

Создание своей ветки

ВАЖНО: называем ветку всегда DATA-****, где **** – номер вашей задачи в Вашем проекте Jira (DATA-2341 и так далее). И никак иначе.

Правила оформления MR

```
cd dwh-scripts
git checkout master
git pull
git checkout -b DATA-****
```

Что здесь происходит:

`git checkout master` – команда “checkout” переключает нас на какую-то ветку, в нашем случае – на мастер, если вдруг мы были не в нем.

`git pull` – подтягиваем изменения из мастера, если вдруг что-то изменилось за это время.

`git checkout -b DATA-****` – создаем новую ветку с названием “DATA-****” и переключаемся на работу в ней.

Сохранение и отправление изменений на сервер

Заходим обратно в исходную папку `dwh-scripts` через терминал. `master`-ветка могла измениться за то время, пока мы работали над изменениями. Подтянем же все изменения:

```
cd dwh-scripts
git checkout master
git pull
git checkout DATA-000
```

Теперь вливаем свои файлы в ветку, сохраняем изменения и отправляем их с локального репозитория на удаленный.

ВАЖНО: комментарий всегда пишем в стиле “DATA-****: бла-бла-бла”.

Где **** – номер вашей задачи в вашем проекте Jira (DATA-2341 и так далее), затем стоит двоеточие и затем – краткий комментарий, что было сделано: “изменена логика поля, убран баг” и тому подобное. **И никак иначе.** If you speak English, a short comment in English will be much appreciated.

```
git add /path/to/file
git status
git commit -m "DATA-****: _"
git push -u origin DWH-****
```

```
(base) yanazhezher@YANAS-MacBook-Pro dwh-scripts % git add .
(base) yanazhezher@YANAS-MacBook-Pro dwh-scripts % git status
On branch DWH-000
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   clickhouse/analytics/addresses.sql

(base) yanazhezher@YANAS-MacBook-Pro dwh-scripts % git commit -m "DWH-000: test MR"
[DWH-000 3d34aa4] DWH-000: test MR
Committer: YANA ZHEZHER <yanazhezher@YANAS-MacBook-Pro.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 2 insertions(+)
(base) yanazhezher@YANAS-MacBook-Pro dwh-scripts % git push
```

Что здесь происходит:

`git add /path/to/file` – тут по очереди перечисляем все файлы, в которых делали изменения и которые хотим влить

`git status` – покажет, где именно произошло изменение. Проверяем, что изменилось ровно то, что мы планировали изменить.

`git commit -m "DATA-000: _"` – делаем комментарий к своему коммиту

`git push -u origin DATA-000` – отправляем коммит на сервер (`git push -u origin DATA-000` нужен в первый раз для каждой новой ветки, `-u origin DATA-000` связывает локальную ветку с веткой на сервере, в последующем делаем просто `git push`)

Создание merge request в интерфейсе гитлаба

```
volchuv@MacBook-Pro-Vladimir Clone_with_SSH %  
volchuv@MacBook-Pro-Vladimir Clone_with_SSH % git push origin DWH-736  
Перечисление объектов: 11, готово.  
Подсчет объектов: 100% (11/11), готово.  
При сжатии изменений используется до 12 потоков  
Сжатие объектов: 100% (6/6), готово.  
Запись объектов: 100% (6/6), 1000 байтов | 1000.00 КиБ/с, готово.  
Всего 6 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакетов 0  
remote:  
remote: To create a merge request for DWH-736, visit:  
remote:   https://gitlab.sbermarket.tech/analytics/datazone/data_lake/dwh-dags/-/merge_requests/new?merge_request%5Bsource_branch%5D=DWH-736  
remote:  
To git.sbermarket.tech:analytics/datazone/data_lake/dwh-dags.git  
* [new branch]      DWH-736 -> DWH-736  
volchuv@MacBook-Pro-Vladimir Clone_with_SSH %
```

После того, как вы сделали `git push`, на экране появится ссылка для создания MR.

Копируем и переходим по ней обратно в gitlab – попадаем на страницу для завершения merge request.

В поле "Description" вносим более расширенный комментарий о том, что было сделано в задаче.

Выбираем "Assignees" Assigned to me (так как вы автор изменений).

Выбираем Reviewers в случае внесения изменений в витрину – владельца (владельцев) витрины.

Labels

Labels

Merge request dependencies

Enter merge request URLs or references (e.g. path/to/project!merge_request_id)

List the merge requests that must be merged before this one.

Approval rules

Approvers

Any eligible user

Approvals required

0

Add approval rule

Reset to project defaults

Tip: add a CODEOWNERS to automatically add approvers based on file paths and file types.

Merge options

☒ Delete source branch when merge request is accepted.

☒ Squash commits when merge request is accepted.

Create merge request

Cancel

Commits 1

Changes 1

28 Oct, 2021 1 commit

DWH-736: change_nominal_cost_in_shipments

c086f429

 Владимир Чувичкин

authored 2 minutes ago

Обязательно проставляем обе галочки: "Delete source branch when merge request is accepted" и "Squash commits when merge request is accepted".

Можно нажимать "Create merge request".

Готово, вы восхитительны!

Все MR-ы будут видны во вкладке "Merge requests" слева в репозитории в интерфейсе (в случае репозитория [dwh-scripts](https://gitlab.sbermarket.tech/analytics/datazone/data_lake/dwh-scripts/-/merge_requests) – https://gitlab.sbermarket.tech/analytics/datazone/data_lake/dwh-scripts/-/merge_requests).



После того, как MR запустился, нужно кинуть ссылку на него в канал [~dwh-support](#). Дежурный дата-инженер посмотрит MR. Когда все замечания учтены, дата-инженер мерджит изменения и запускает скрипт для изменения DDL. Затем ждите свои изменения по расписанию обновления таблицы.

Прочие полезные команды

`git branch -d DATA-***` – удалить локально со своего компьютера

`git branch -D DATA-***` – удалить ветку вообще совсем из репозитория

`git branch -` посмотреть все созданные локально тобой ветки

`git branch -r` – посмотреть вообще все ветки данного репозитория, включая сделанные кем-то еще