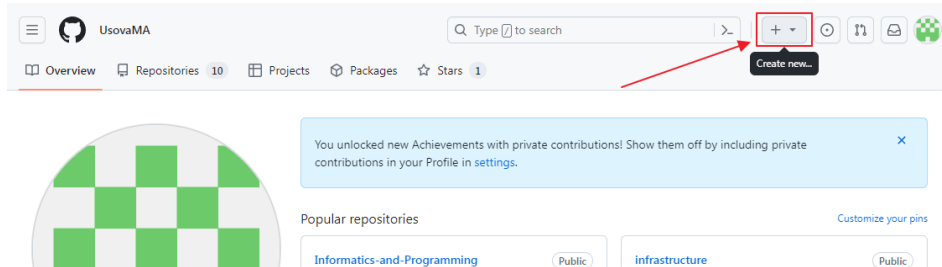


ИНСТРУКЦИЯ ПО GIT И GITHUB

Первое использование / новый репозиторий

1 шаг. Создать аккаунт на [GitHub](#). Скачать и установить [Git BASH](#).

2 шаг. Создайте на GitHub новый репозиторий.



В окне создания репозитория пропишите качественное название и описание. Проставьте необходимые галочки.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Repository template

Start your repository with a template repository's contents.

Owner * / **Repository name ***

✓ OOP is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-dollop](#) ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

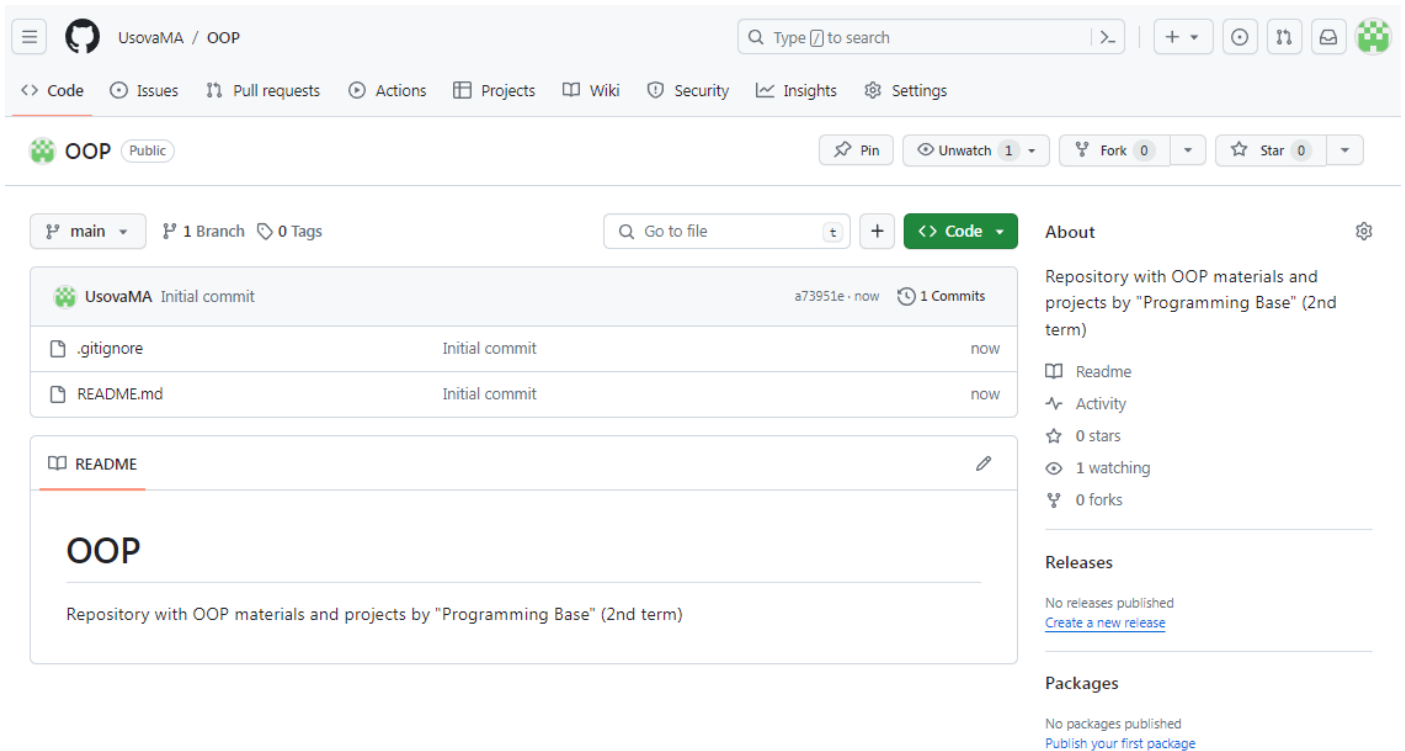
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

ⓘ You are creating a public repository in your personal account.

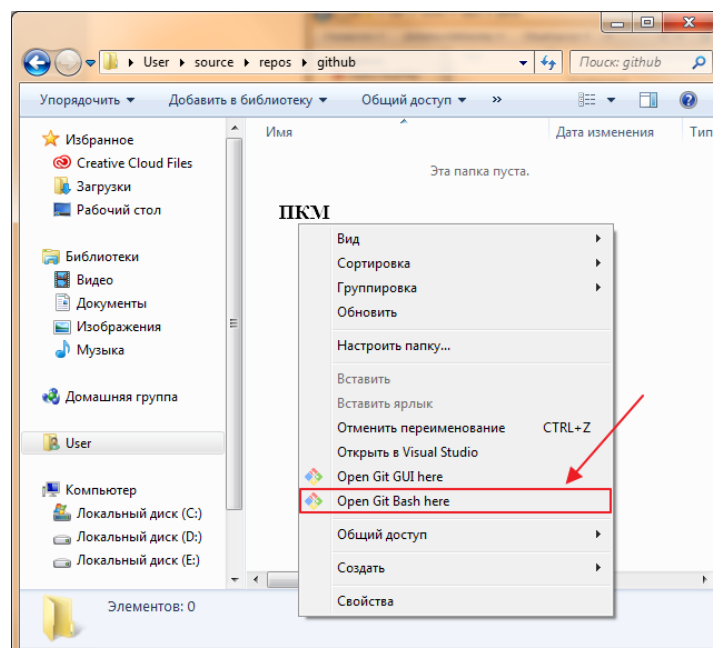
В результате будет создан новый репозиторий с 2 файлами:

- `README.md` – необходим для оформления описания к вашему репозиторию,
- `.gitignore` – содержит список файлов, которые не должны подтягиваться на GitHub.



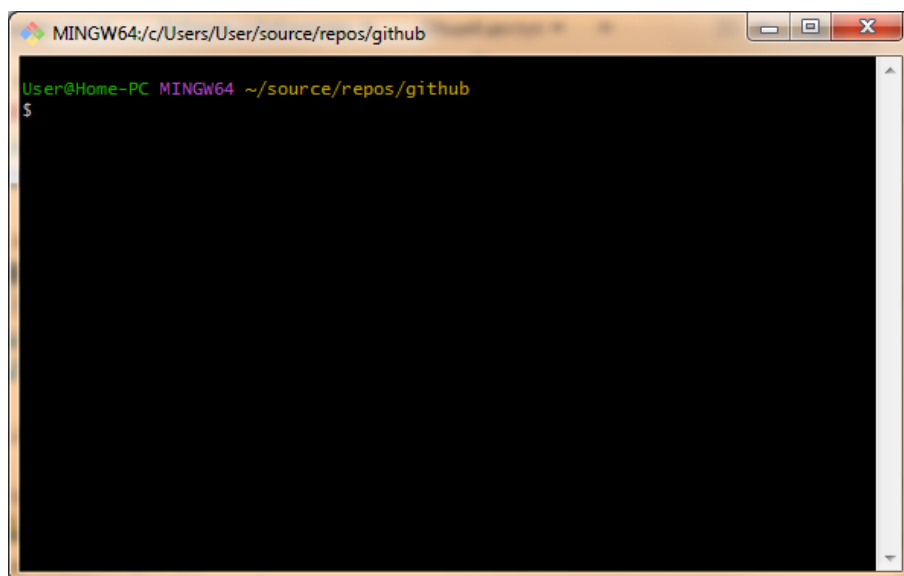
3 шаг. Создайте у себя на компьютере локальную копию этого репозитория.

1. Выберите расположение для ваших github-проектов. Убедитесь, что в пути нет русских символов и пробелов.
2. Щелкните ПКМ¹ по свободному пространству. В появившемся списке свойств вызовите `git bash`.

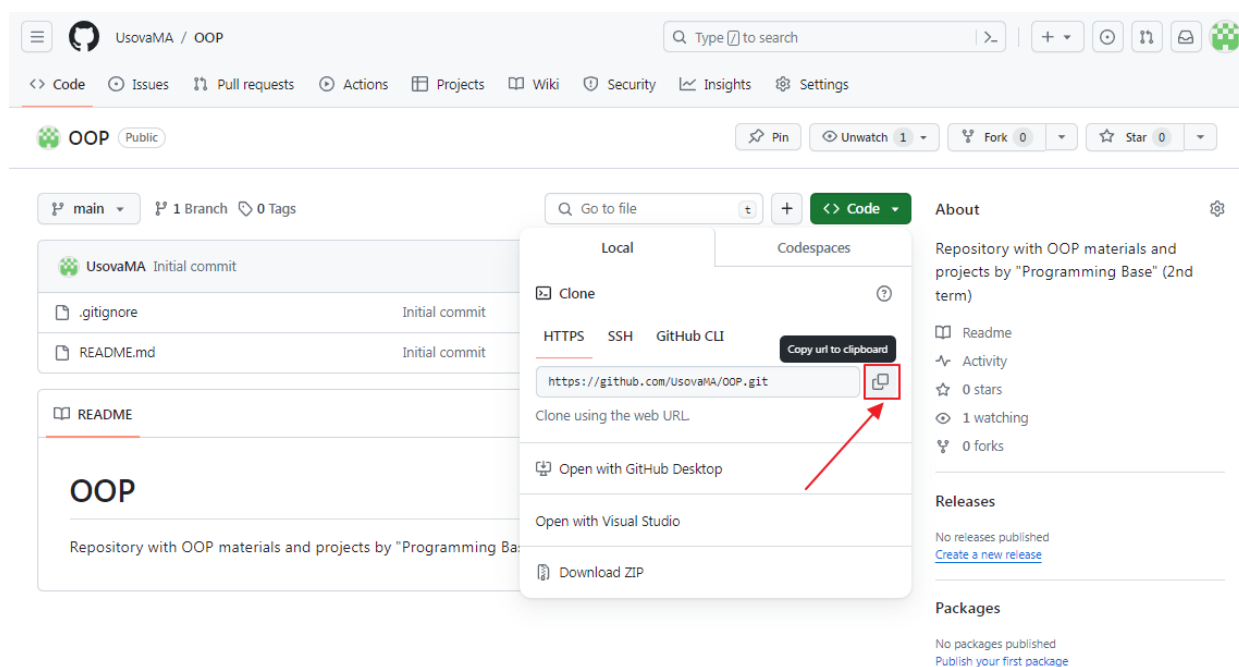


¹ ПКМ – здесь и далее означает «правая кнопка мыши».

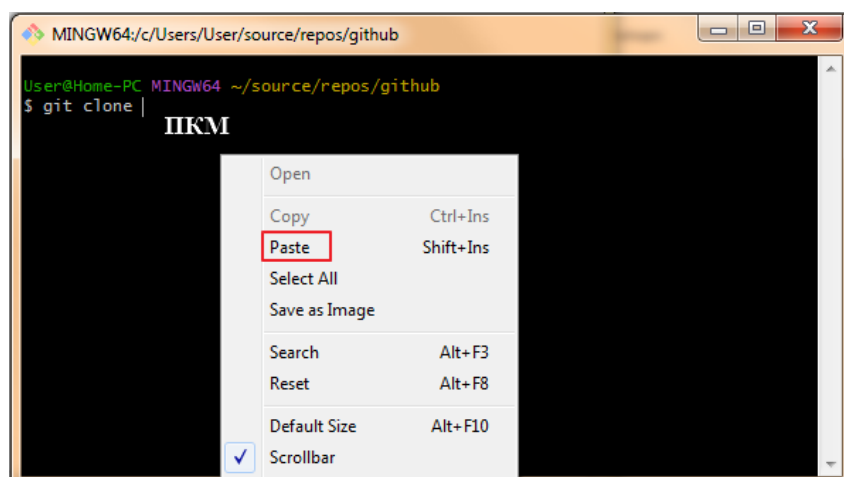
Откроется консоль, с которой нам и предстоит работать.



3. Скопируйте ссылку на ваш созданный репозиторий с GitHub.



4. В консоли наберите команду `git clone` и добавьте скопированную ссылку на ваш репозиторий. Затем выполните команду, нажав на enter.



После того как проект скачается, перейдите в появившуюся паку с помощью команды `cd`, указав название папки (название репозитория):

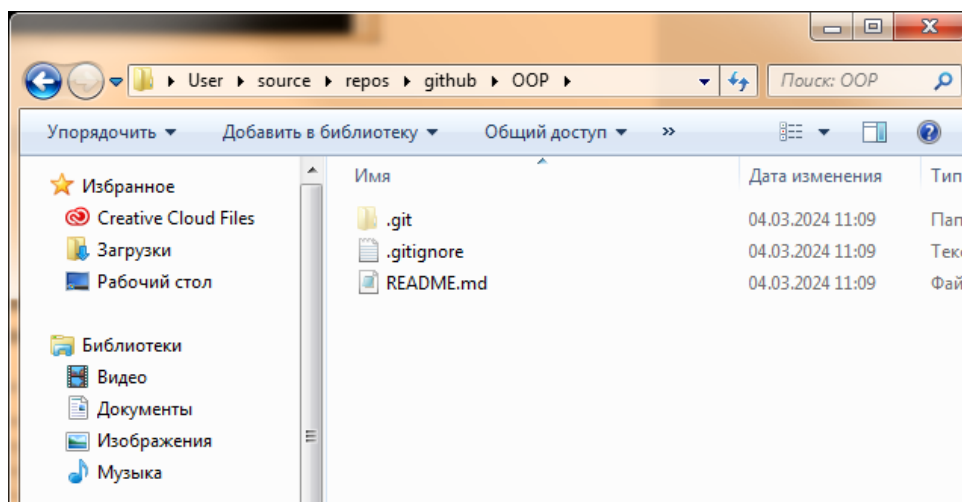
```
MINGW64/c/Users/User/source/repos/github/OOP

User@Home-PC MINGW64 ~/source/repos/github
$ git clone https://github.com/UsovaMA/OOP.git
Cloning into 'OOP'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

User@Home-PC MINGW64 ~/source/repos/github
$ cd OOP

User@Home-PC MINGW64 ~/source/repos/github/OOP (main)
$ |
```

Состояние папки соответствует состоянию вашего клонированного репозитория:



4 шаг. Создайте с помощью команды `git branch` (с указанием названия) новую ветку разработки. Перейдите в эту созданную ветку с помощью команды `git checkout` (с указанием названия).

```
MINGW64/c/Users/User/source/repos/github/OOP

User@Home-PC MINGW64 ~/source/repos/github
$ git clone https://github.com/UsovaMA/OOP.git
Cloning into 'OOP'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

User@Home-PC MINGW64 ~/source/repos/github
$ cd OOP

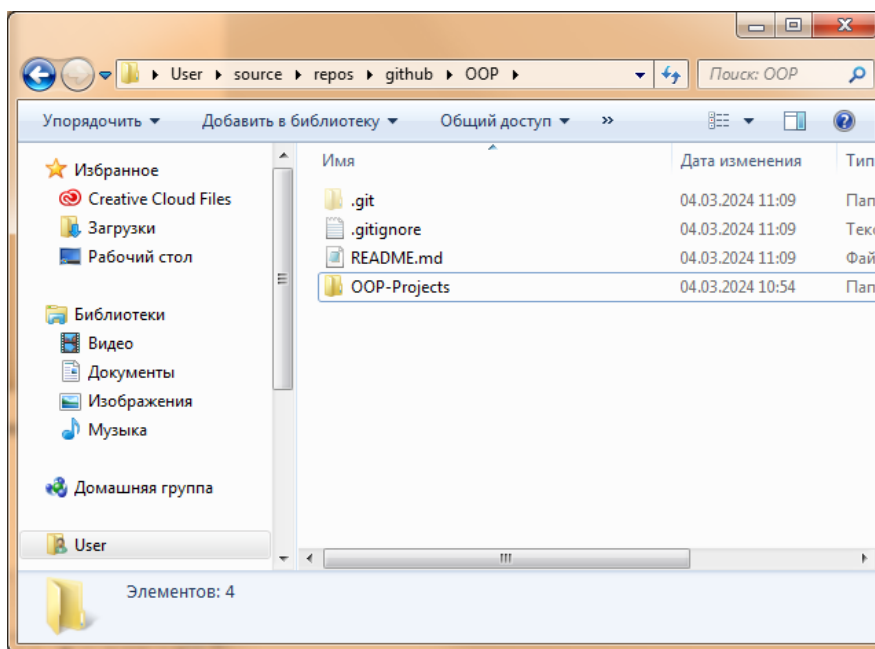
User@Home-PC MINGW64 ~/source/repos/github/OOP (main)
$ git branch implementation-time

User@Home-PC MINGW64 ~/source/repos/github/OOP (main)
$ git checkout implementation-time
Switched to branch 'implementation-time'

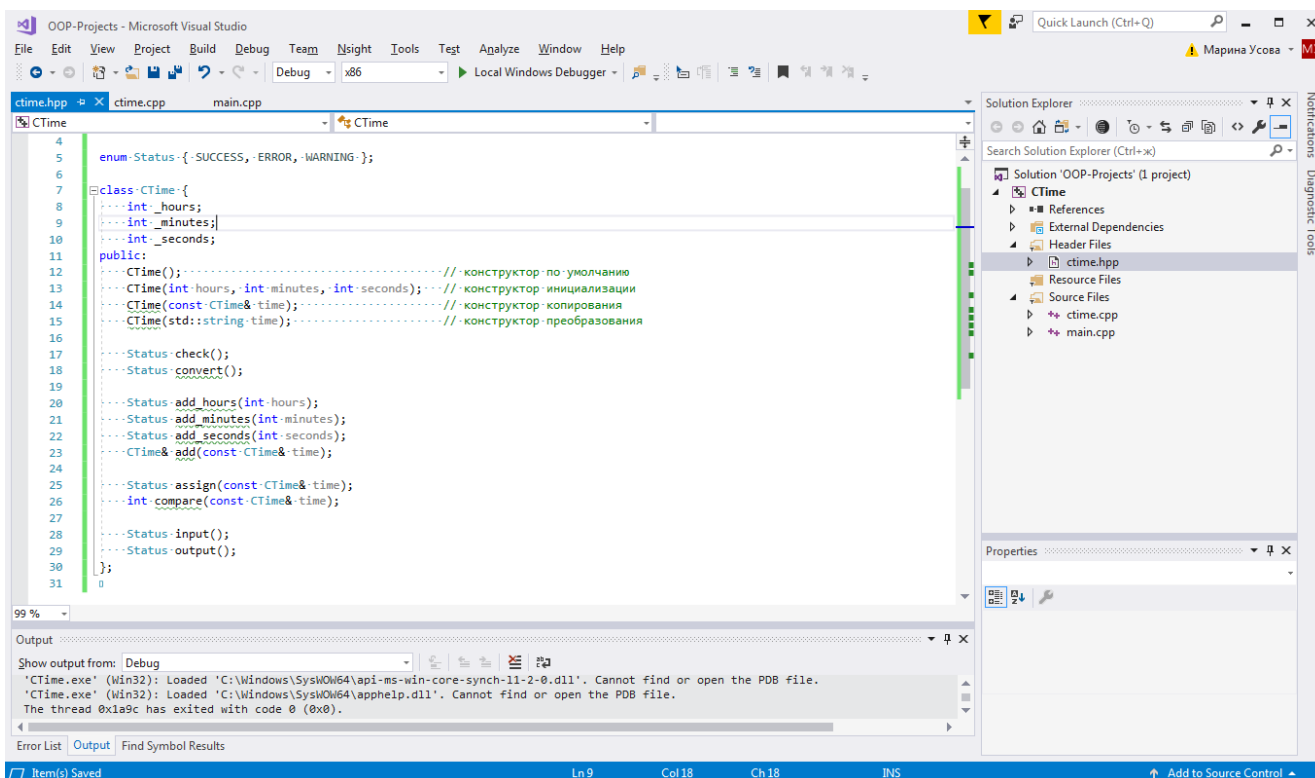
User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$
```

Теперь это ваша рабочая ветка для текущей рабочей задачи.

5 шаг. Создайте решение. Например, для выполнения работ в данном семестре (OOP-Projects) с первым проектом (CTime). При создании расположите его в клонированном (локальном) репозитории.



Выполните набросок вашей текущей работы (создайте все необходимые файлы, добавьте описания классов с примерным набором функционала).

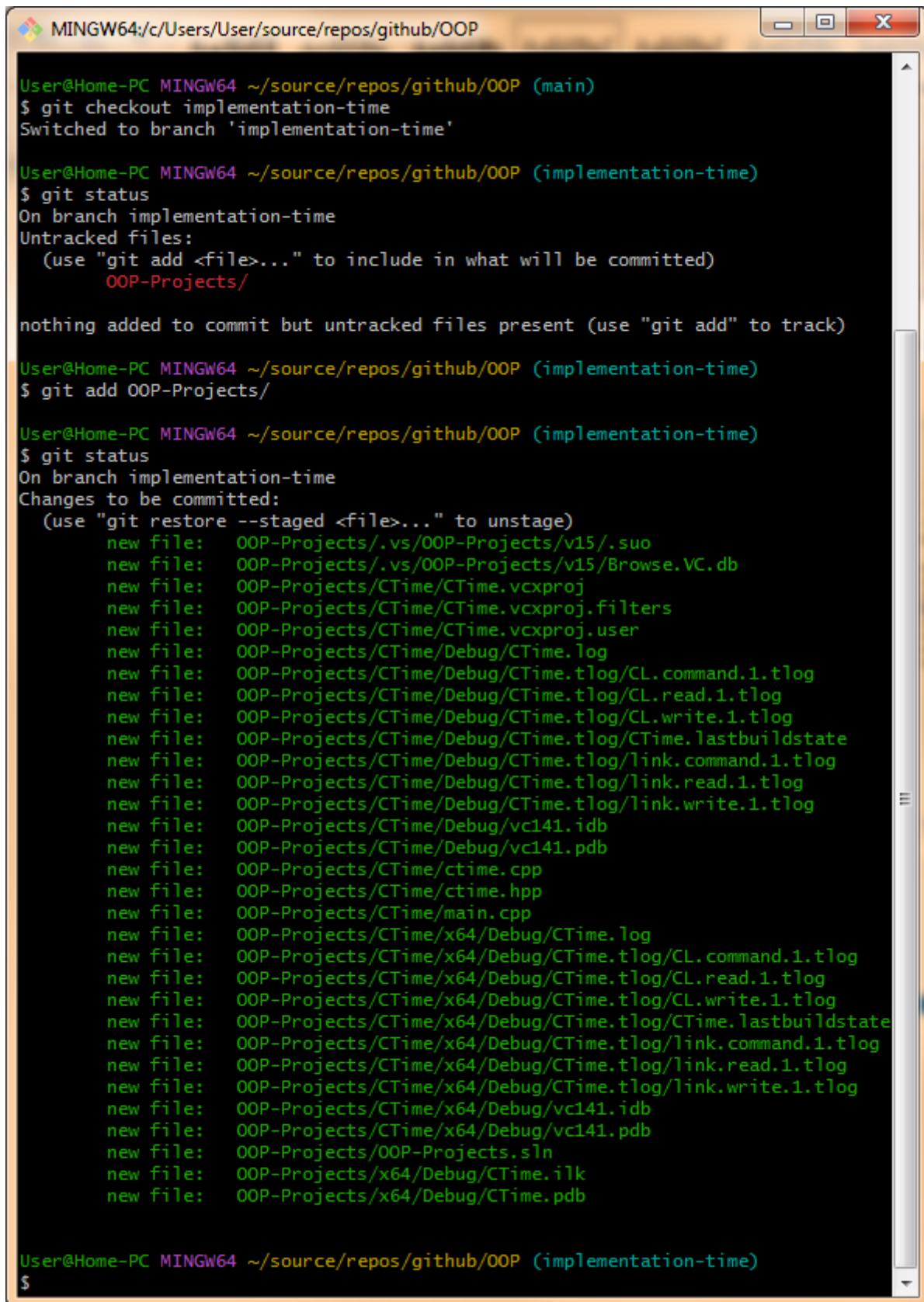


6 шаг. Выложите этот набросок на GitHub.

1. Проверьте состояние репозитория с помощью команды `git status`.
2. Добавьте необходимые файлы к будущему коммиту (точке сохранения для вашего проекта, к которой в любой момент можно вернуться) с помощью команды `git add` с указанием добавляемых файлов.
3. Проверьте состояние будущего коммита с помощью команды `git status`.

4. В результате могут быть добавлены лишние файлы, которым на GitHub не место (они не влияют на сборку проекта, но весят очень много). Требуется устранить эти файлы и поправить в `.gitignore` игнорируемые папки сборки.

Например, в моём случае это `.vs/`, `Debug/`, `x64/`:



```
MINGW64:/c/Users/User/source/repos/github/OOP

User@Home-PC MINGW64 ~/source/repos/github/OOP (main)
$ git checkout implementation-time
Switched to branch 'implementation-time'

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git status
On branch implementation-time
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    OOP-Projects/

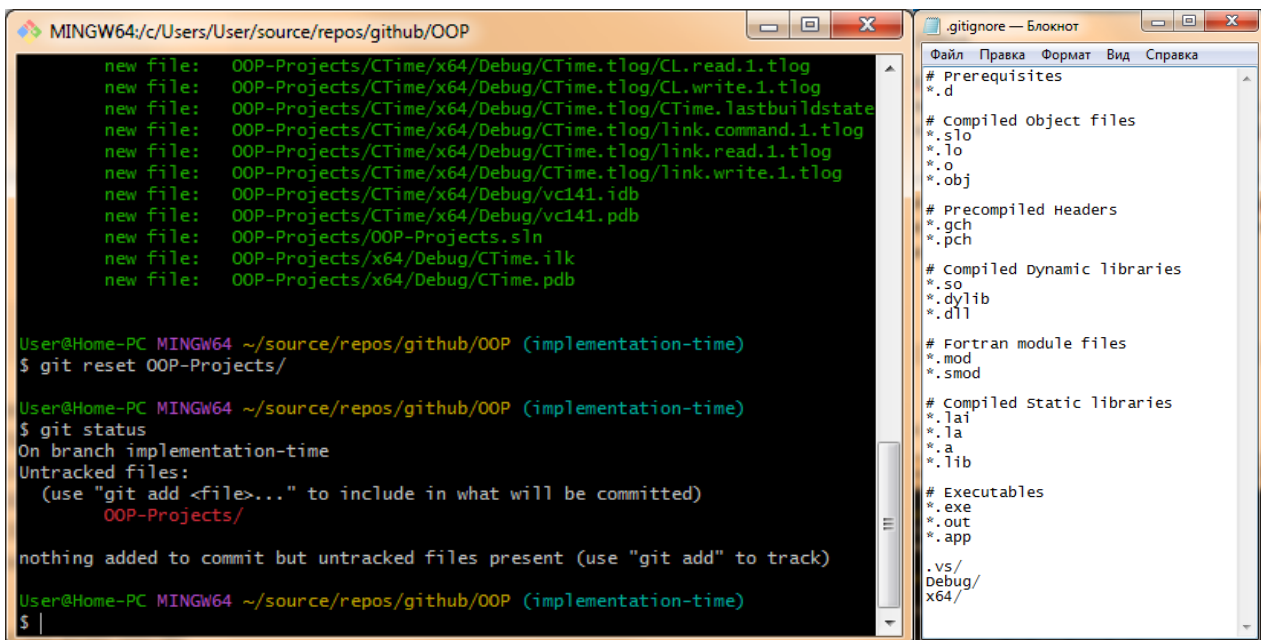
nothing added to commit but untracked files present (use "git add" to track)

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git add OOP-Projects/

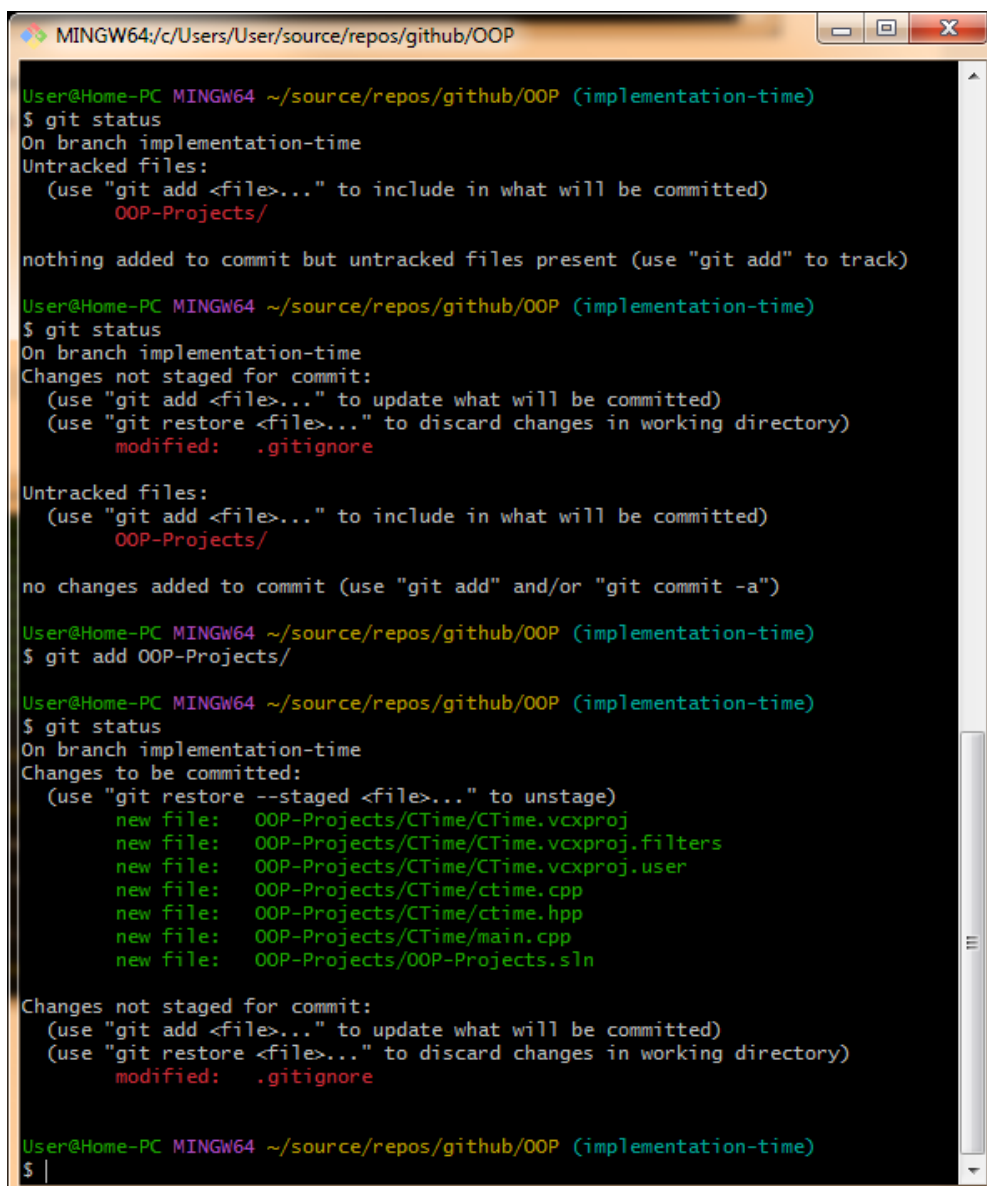
User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git status
On branch implementation-time
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   OOP-Projects/.vs/OOP-Projects/v15/.suo
    new file:   OOP-Projects/.vs/OOP-Projects/v15/Browse.VC.db
    new file:   OOP-Projects/CTime/CTime.vcxproj
    new file:   OOP-Projects/CTime/CTime.vcxproj.filters
    new file:   OOP-Projects/CTime/CTime.vcxproj.user
    new file:   OOP-Projects/CTime/Debug/CTime.log
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/CL.command.1.tlog
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/CL.read.1.tlog
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/CL.write.1.tlog
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/CTime.lastbuildstate
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/link.command.1.tlog
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/link.read.1.tlog
    new file:   OOP-Projects/CTime/Debug/CTime.tlog/link.write.1.tlog
    new file:   OOP-Projects/CTime/Debug/vc141.idb
    new file:   OOP-Projects/CTime/Debug/vc141.pdb
    new file:   OOP-Projects/CTime/ctime.cpp
    new file:   OOP-Projects/CTime/ctime.hpp
    new file:   OOP-Projects/CTime/main.cpp
    new file:   OOP-Projects/CTime/x64/Debug/CTime.log
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/CL.command.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/CL.read.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/CL.write.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/CTime.lastbuildstate
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/link.command.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/link.read.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/CTime.tlog/link.write.1.tlog
    new file:   OOP-Projects/CTime/x64/Debug/vc141.idb
    new file:   OOP-Projects/CTime/x64/Debug/vc141.pdb
    new file:   OOP-Projects/OOP-Projects.sln
    new file:   OOP-Projects/x64/Debug/CTime.ilc
    new file:   OOP-Projects/x64/Debug/CTime.pdb

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$
```

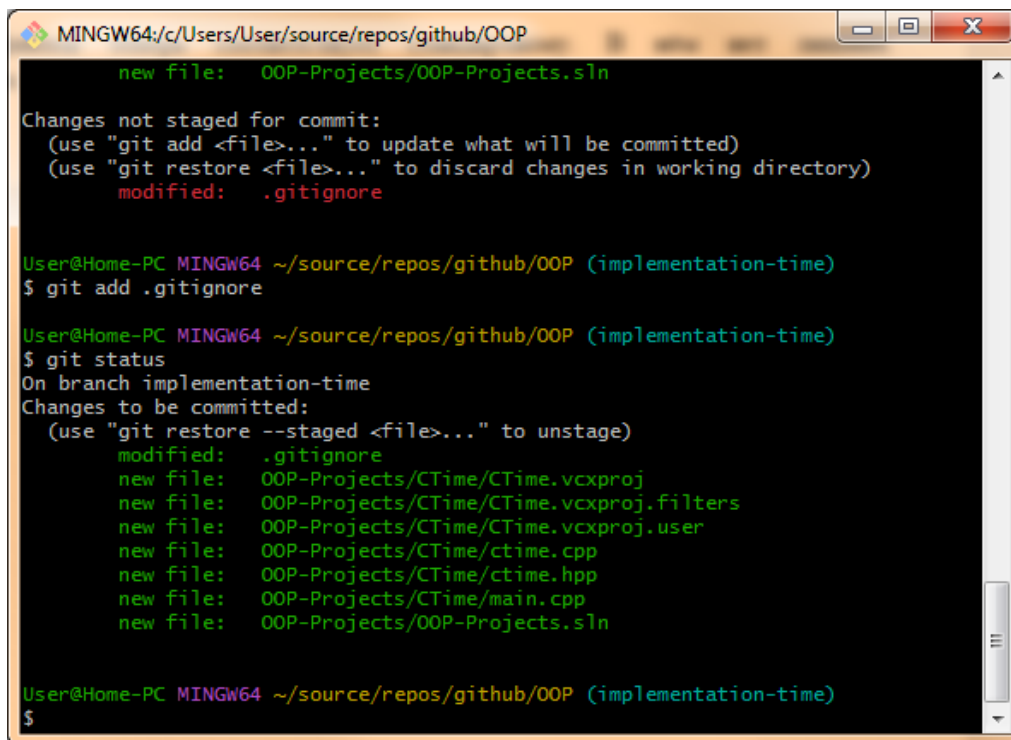
Для отмены этого неудачного коммита использовать команду `git reset` с указанием отклоняемых от коммита файлов. В нашем случае проще отменить всю папку и поправить файл `.gitignore`:



После этого повторяем попытку добавить всю папку к коммиту:



Список добавляемых файлов теперь соответствует стандартному. В нём нет лишних конфигурационных файлов. Также к коммиту следует добавить изменённый файл `.gitignore`:



```
MINGW64/c/Users/User/source/repos/github/OOP

new file:   OOP-Projects/OOP-Projects.sln

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:   .gitignore

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git add .gitignore

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git status
On branch implementation-time
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   OOP-Projects/CTime/CTime.vcxproj
        new file:   OOP-Projects/CTime/CTime.vcxproj.filters
        new file:   OOP-Projects/CTime/CTime.vcxproj.user
        new file:   OOP-Projects/CTime/ctime.cpp
        new file:   OOP-Projects/CTime/ctime.hpp
        new file:   OOP-Projects/CTime/main.cpp
        new file:   OOP-Projects/OOP-Projects.sln

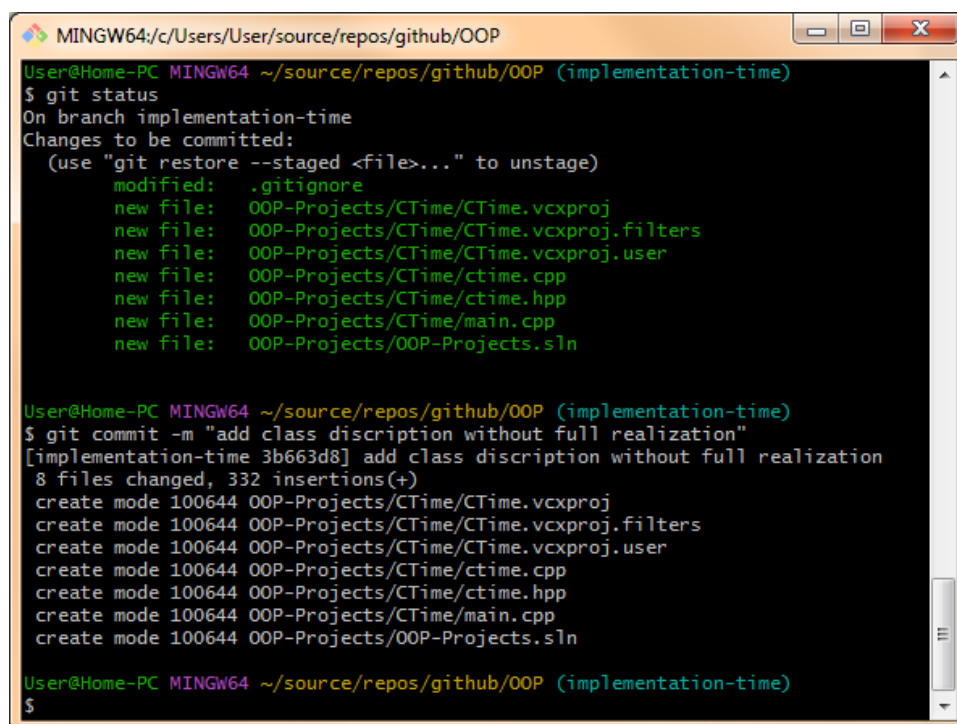
User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$
```

Совет: чаще прожимайте `git status`, чтобы проверить состояние будущего коммита.

5. Создайте коммит с помощью команды

```
git commit -m "comment to this checkpoint, what's done?"
```

Давайте **качественные** комментарии вашим коммитам, чтобы было ясно, что по сравнению с прошлой версией кода было сделано.



```
MINGW64/c/Users/User/source/repos/github/OOP

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git status
On branch implementation-time
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore
        new file:   OOP-Projects/CTime/CTime.vcxproj
        new file:   OOP-Projects/CTime/CTime.vcxproj.filters
        new file:   OOP-Projects/CTime/CTime.vcxproj.user
        new file:   OOP-Projects/CTime/ctime.cpp
        new file:   OOP-Projects/CTime/ctime.hpp
        new file:   OOP-Projects/CTime/main.cpp
        new file:   OOP-Projects/OOP-Projects.sln

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git commit -m "add class discription without full realization"
[implementation-time 3b663d8] add class discription without full realization
 8 files changed, 332 insertions(+)
 create mode 100644 OOP-Projects/CTime/CTime.vcxproj
 create mode 100644 OOP-Projects/CTime/CTime.vcxproj.filters
 create mode 100644 OOP-Projects/CTime/CTime.vcxproj.user
 create mode 100644 OOP-Projects/CTime/ctime.cpp
 create mode 100644 OOP-Projects/CTime/ctime.hpp
 create mode 100644 OOP-Projects/CTime/main.cpp
 create mode 100644 OOP-Projects/OOP-Projects.sln

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$
```

6. Отправьте коммит на GitHub с помощью команды `git push origin` с указанием названия рабочей ветки.


```
MINGW64/c/Users/User/source/repos/github/OOP
User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git push origin implementation-time
fatal: Отказано в доступе
error: unable to read askpass response from 'C:/Program Files/Git/mingw64/bin/git-askpass.exe'
Username for 'https://github.com': UsovaMA
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/UsovaMA/OOP.git/'

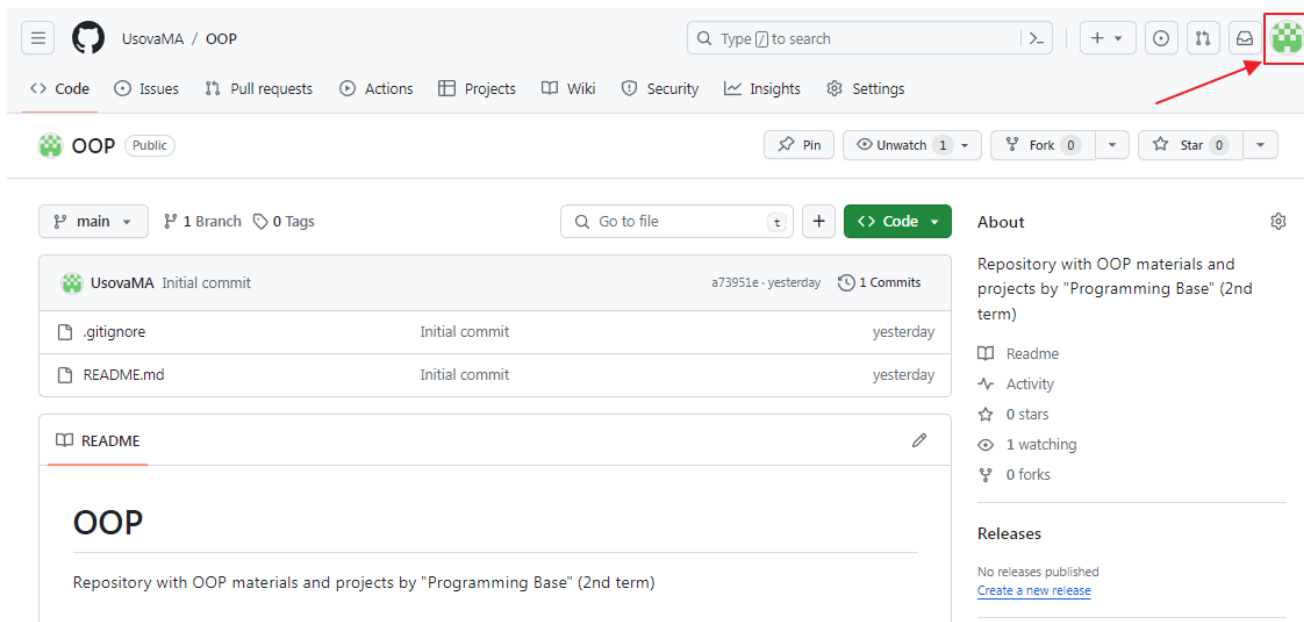
User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ |
```

Примечание. Рекомендовано в принципе после выполнения любой команды читать сообщения, которые пишет git. **Fatal** и **error** сообщения (как в примере выше) означают, что команда не выполнялась и нужно решать некоторую проблему. **Не выполняйте другие действия привычной последовательности, пока не будет решена проблема!!! Исправить серию ошибочного вызова команд сложнее одной текущей!!!** В консоли у нас есть информация об ошибках, их необходимо читать и переводить. Если перевод не помог понять, что необходимо сделать – ошибку следует загуглить. Часто при выводе ошибок, git указывает способы решения, рекомендовано пробовать выполнять команды, которые предлагает git.

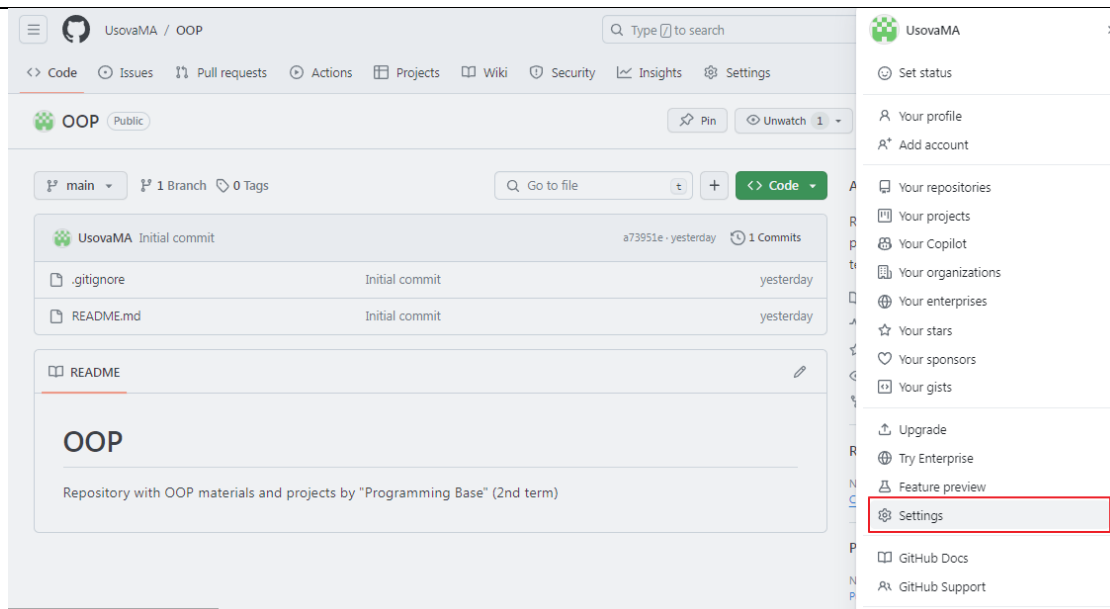
В данном случае проблема в авторизации. В интернете можно наткнуться на новость, что «Git support for password authentication was removed» и решение проблемы заключается в использовании защищенного подключения и token.

СОЗДАНИЕ ТОКЕНА

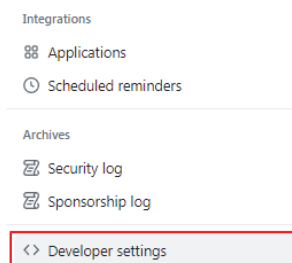
1. Открыть настройки профиля:



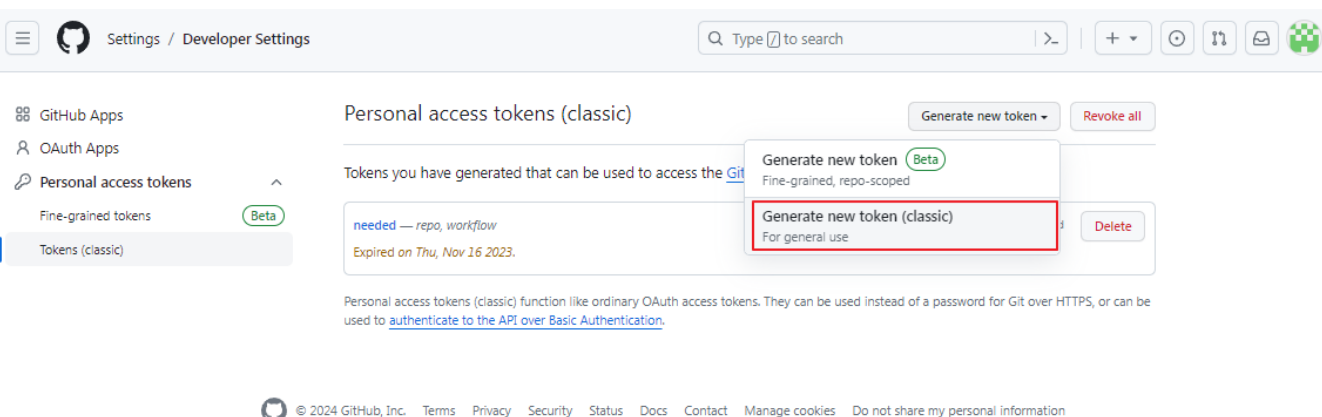
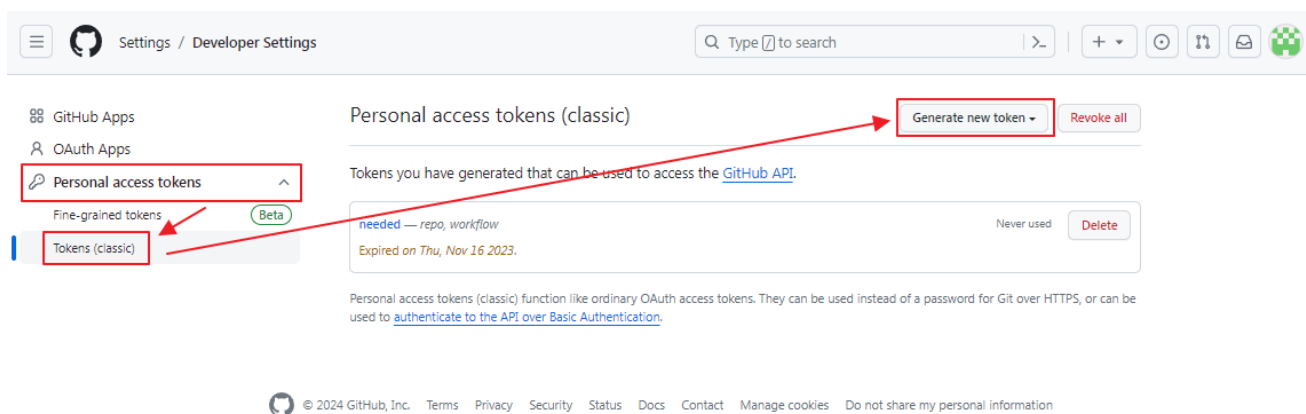
The screenshot shows the GitHub interface for the repository 'UsovaMA / OOP'. In the top navigation bar, the 'Settings' icon (a gear) is highlighted with a red box and a red arrow. Below the repository name, there are tabs for 'main', '1 Branch', and '0 Tags'. The main content area shows the repository's commit history with files like '.gitignore' and 'README.md'. The right sidebar contains repository statistics and links to 'About', 'Releases', and 'Packages'.



2. В конце страницы открывшихся настроек находим Developer settings:



3. Сгенерировать токен:



Произвести настройки токена:

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Note

needed for push with authentication

What's this token for?

Expiration *

90 days

The token will expire on Mon, Jun 3 2024

1. позже при необходимости пересоздаёте token;
2. можно проставить все галочки.

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input checked="" type="checkbox"/> workflow	Update GitHub Action workflows
<input checked="" type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input checked="" type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input checked="" type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input checked="" type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input checked="" type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input checked="" type="checkbox"/> read:org	Read org and team membership, read org projects
<input checked="" type="checkbox"/> manage_runners:org	Manage org runners and runner groups
<input checked="" type="checkbox"/> admin:public_key	Full control of user public keys
<input checked="" type="checkbox"/> write:public_key	Write user public keys

4. После создания токена на экране появится его код. **Его нужно сохранить и иметь всегда к нему доступ!**

Personal access tokens (classic)

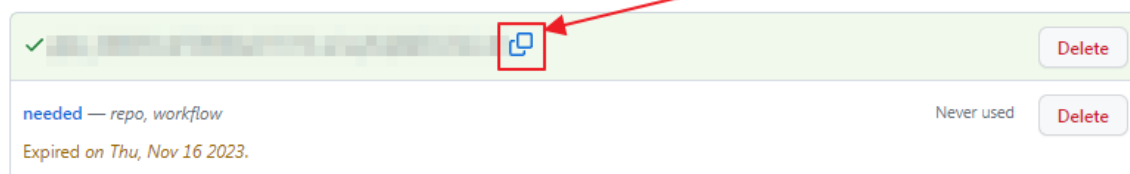
Generate new token ▼

Revoke all

Tokens you have generated that can be used to access the [GitHub API](#).

скопировать и
сохранить
себе!

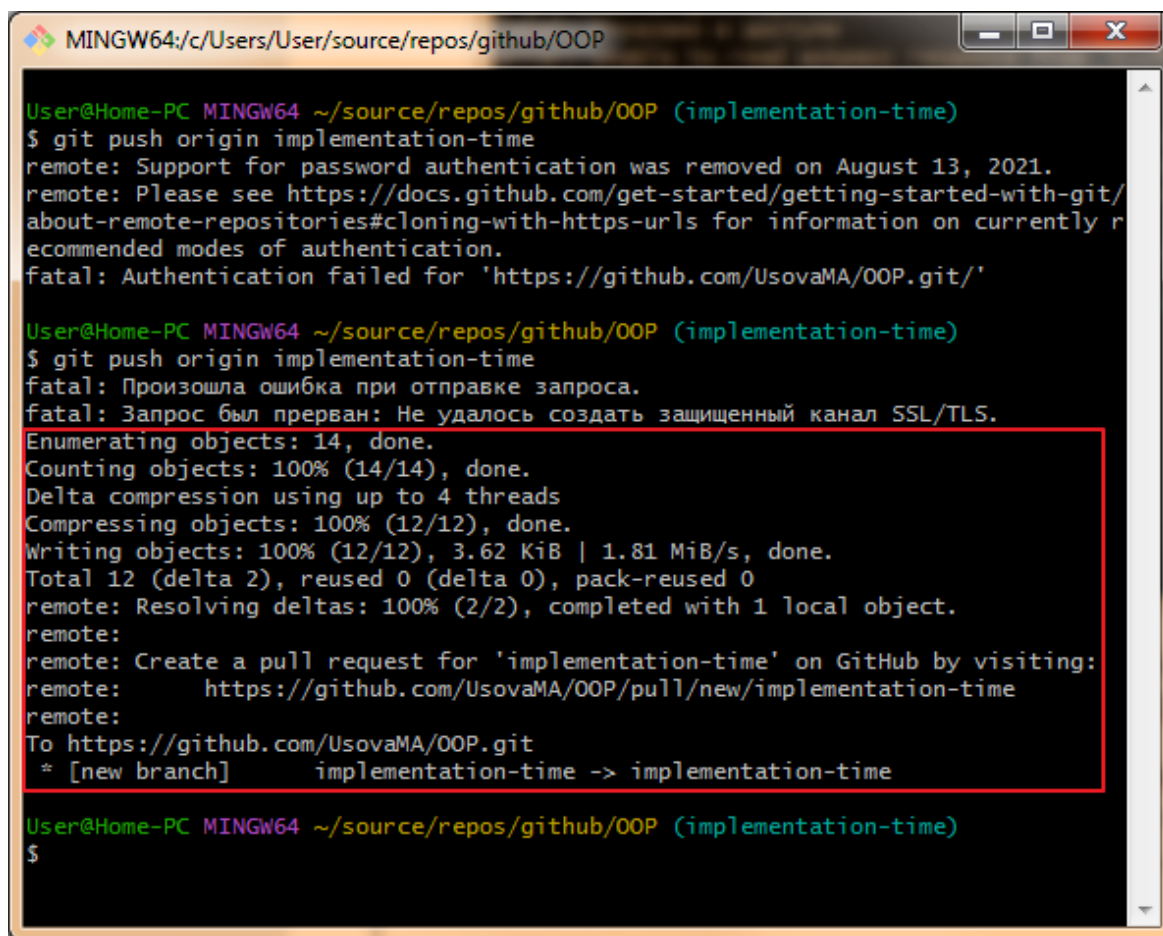
Make sure to copy your personal access token now. You won't be able to see it again!



Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

5. Теперь вместо пароля при авторизации использовать этот токен.

После создания токена (делается **один раз** на долгий период, а не при каждом push) повторить попытку отправить код на GitHub с помощью команды `git push origin` с указанием названия ветки. В качестве пароля теперь вводится сгенерированный токен!



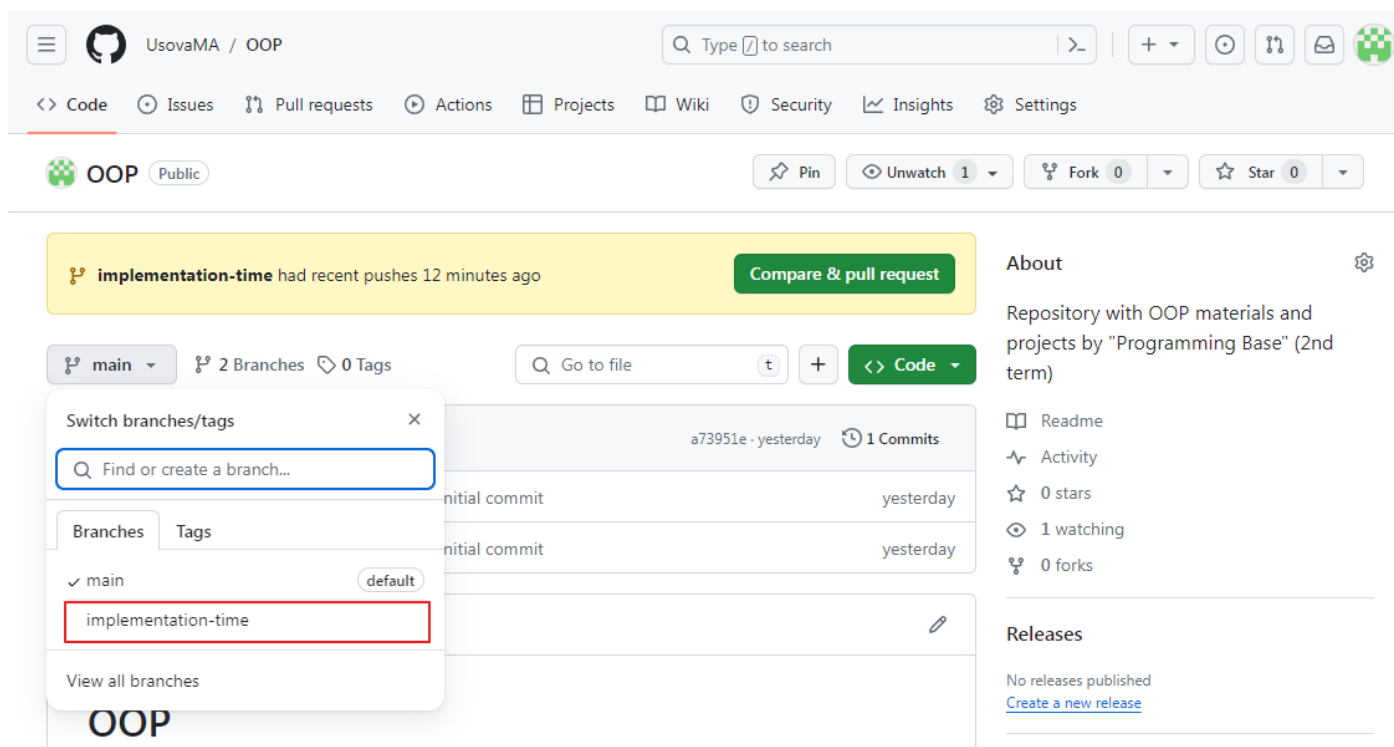
```
MINGW64:/c/Users/User/source/repos/github/OOP

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git push origin implementation-time
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/UsovaMA/OOP.git/'

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$ git push origin implementation-time
fatal: Произошла ошибка при отправке запроса.
fatal: Запрос был прерван: Не удалось создать защищенный канал SSL/TLS.
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 3.62 KiB | 1.81 MiB/s, done.
Total 12 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
remote:
remote: Create a pull request for 'implementation-time' on GitHub by visiting:
remote:   https://github.com/UsovaMA/OOP/pull/new/implementation-time
remote:
To https://github.com/UsovaMA/OOP.git
 * [new branch]      implementation-time -> implementation-time

User@Home-PC MINGW64 ~/source/repos/github/OOP (implementation-time)
$
```

Код отправлен на GitHub в новую создавшуюся ветку:



Совет: проверяйте, что код действительно отправился на GitHub, и вы не проглядели где-то в процессе работы ошибки типа fatal.

7 шаг. Разработка.

Далее разработка включает написание оставшихся функций, создание серии коммитов и выкладывание их на GitHub. Коммитом может быть в том числе небольшое исправление бага. Нормальная история коммитов должна хорошо комментировать историю разработки.

8 шаг. Ревью кода и релиз.

Когда код по мнению разработчика готов, он создаёт pull request и приглашает на ревью коллег / тимлида / преподавателя.

После создания, следует написать комментарий с обращением к ревьюерам с просьбой одобрить код.

The screenshot shows a GitHub Pull Request (PR) titled "Разработка класса Время #1" (Development of the Time class #1). The PR is open, showing a commit from the "implementation-time" branch to the "main" branch. The commit message is "было добавлено объявление класса времени и реализация" (the declaration of the time class and implementation was added). The PR is currently in progress, with no reviews. The interface includes a "Merge pull request" button and a "Comment" button. The comment box contains the text "@UsovaMA готово" (@UsovaMA is ready). The right sidebar shows the "Reviewers" section with "No reviews" and the "Assignees" section with "No one—assign yourself". The bottom of the interface includes a "Close with comment" button and a "Comment" button.

По мере замечаний ревьюеров, как обычно правим код и заливаем коммиты. Они автоматически будут подтягиваться в этот pull request. Как только одобрение ревьюеров получено, можно выполнить «релиз» выкладыванием кода из рабочей ветки в основную (main).

Примечание 1. Подробнее о создании pull request и процессе общения с ревьюерами в отдельном файле.

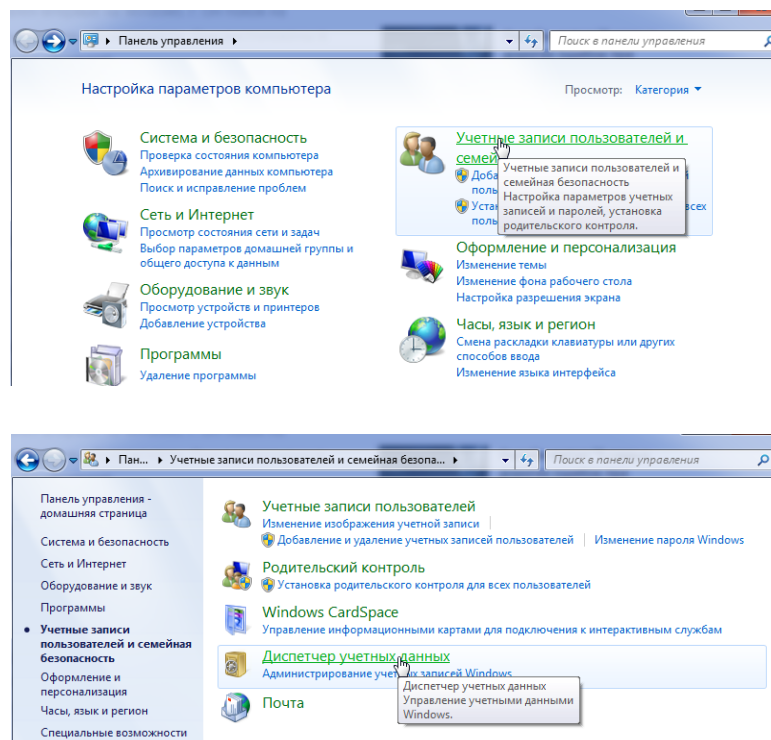
Примечание 2. Список основных и просто полезных команд в отдельном файле.

КАК СДЕЛАТЬ АВТОМАТИЧЕСКУЮ АВТОРИЗАЦИЮ ПРИ PUSH (только в Windows)

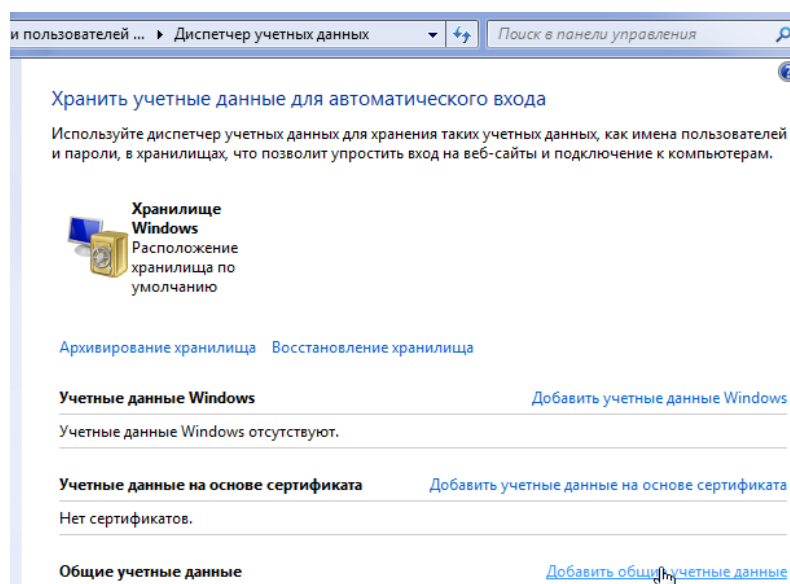
ТОЛЬКО НА СВОЁМ ЛИЧНОМ КОМПЬЮЕРЕ

В УНИВЕРСИТЕТЕ, ЕСЛИ ВЫ СИДИТЕ ПОД ОДНИМ ЛОГИНОМ С ДРУГИМ ЧЕЛОВЕКОМ,
ОН БУДЕТ ЭТУ ЗАПИСЬ УДАЛЯТЬ КАЖДЫЙ РАЗ

1. Панель Управления -> Учетные записи пользователей -> Диспетчер учетных данных (или Credential Manager).



2. Далее нажимаем в разделе Общие учетные данные — Добавить общие учетные данные



3. Вводим — адрес сайта GitHub с префиксом git:, имя пользователя как на GitHub, а в поле пароль **вводим не сам пароль от аккаунта, а токен**, который мы до этого создали и жмем ОК.

Введите адрес веб-сайта или сетевое размещение и учетные данные

Убедитесь, что введенное имя пользователя и пароль могут использоваться для доступа к размещению.

Адрес в Интернете или сети:

Имя пользователя:

Пароль: