

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

**«Исследование основных возможностей Git и GitHub»
Отчет по лабораторной работе № 1.1
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-22-1
Душин Александр Владимирович.
Подпись студента _____
Работа защищена « » _____ 20__ г.
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

作業目標: Git バージョンコントロールシステムと IT プロジェクトのホスティングウェブサービス **GitHub** の基本機能を調査します。

進行中の作業:

1. MIT ライセンスが使用され、選択したプログラミング言語が含まれる **GitHub** 上のパブリックリポジトリを作成します。

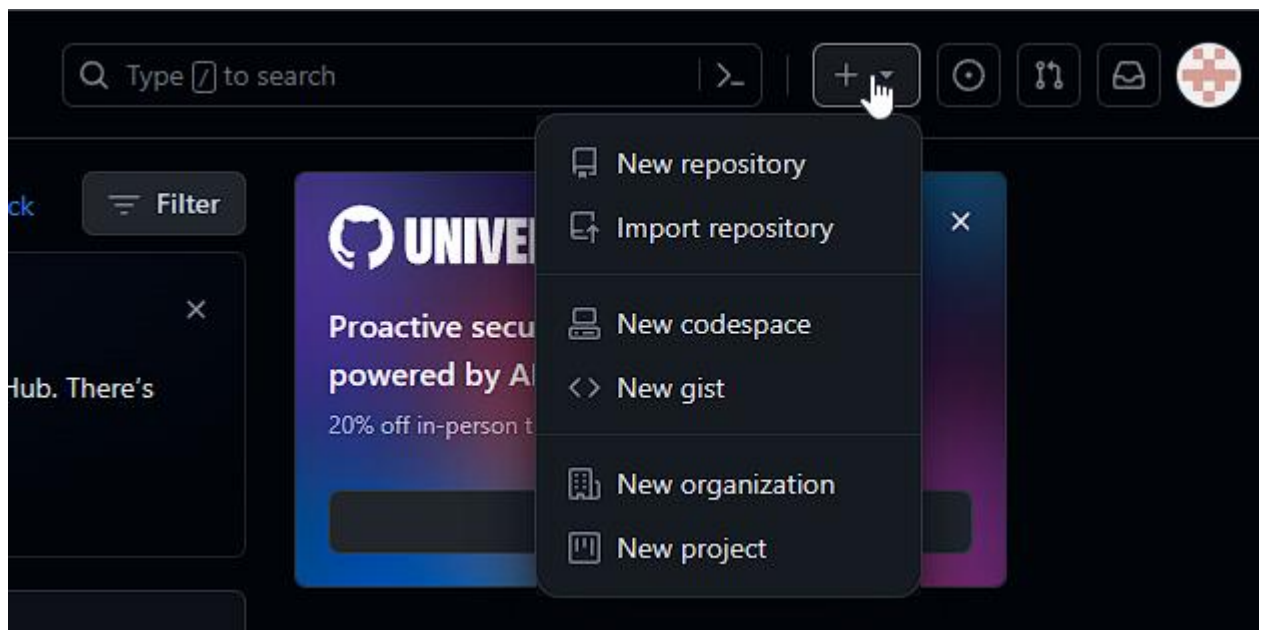


図1 – 新しいリポジトリの作成

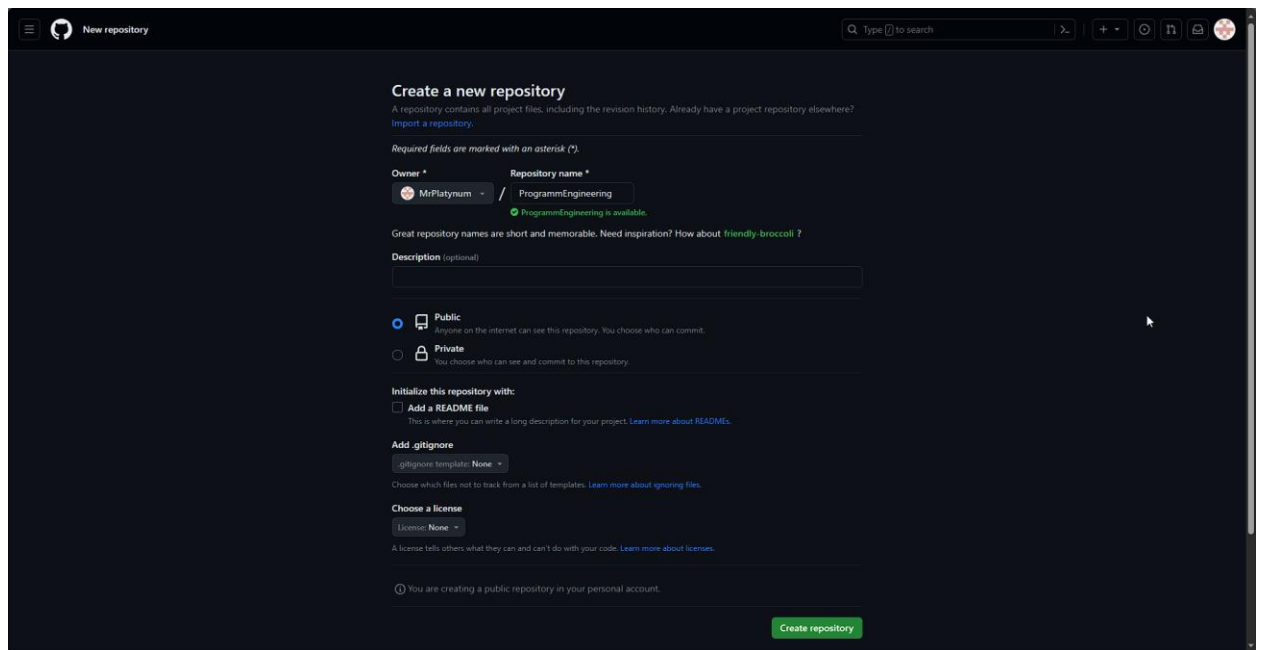


図2 – 新しいリポジトリの作成ページ

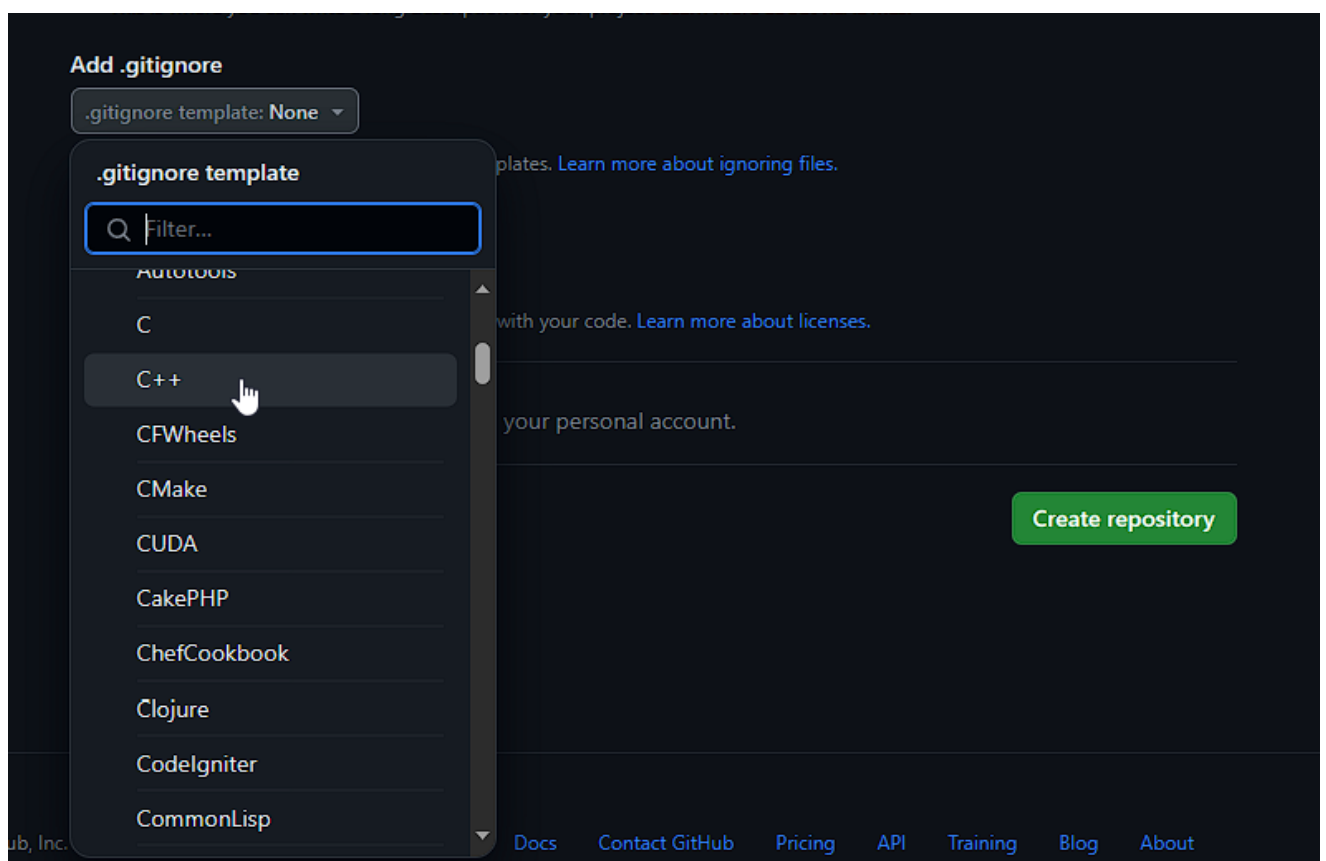


図3 – プログラム言語の選択

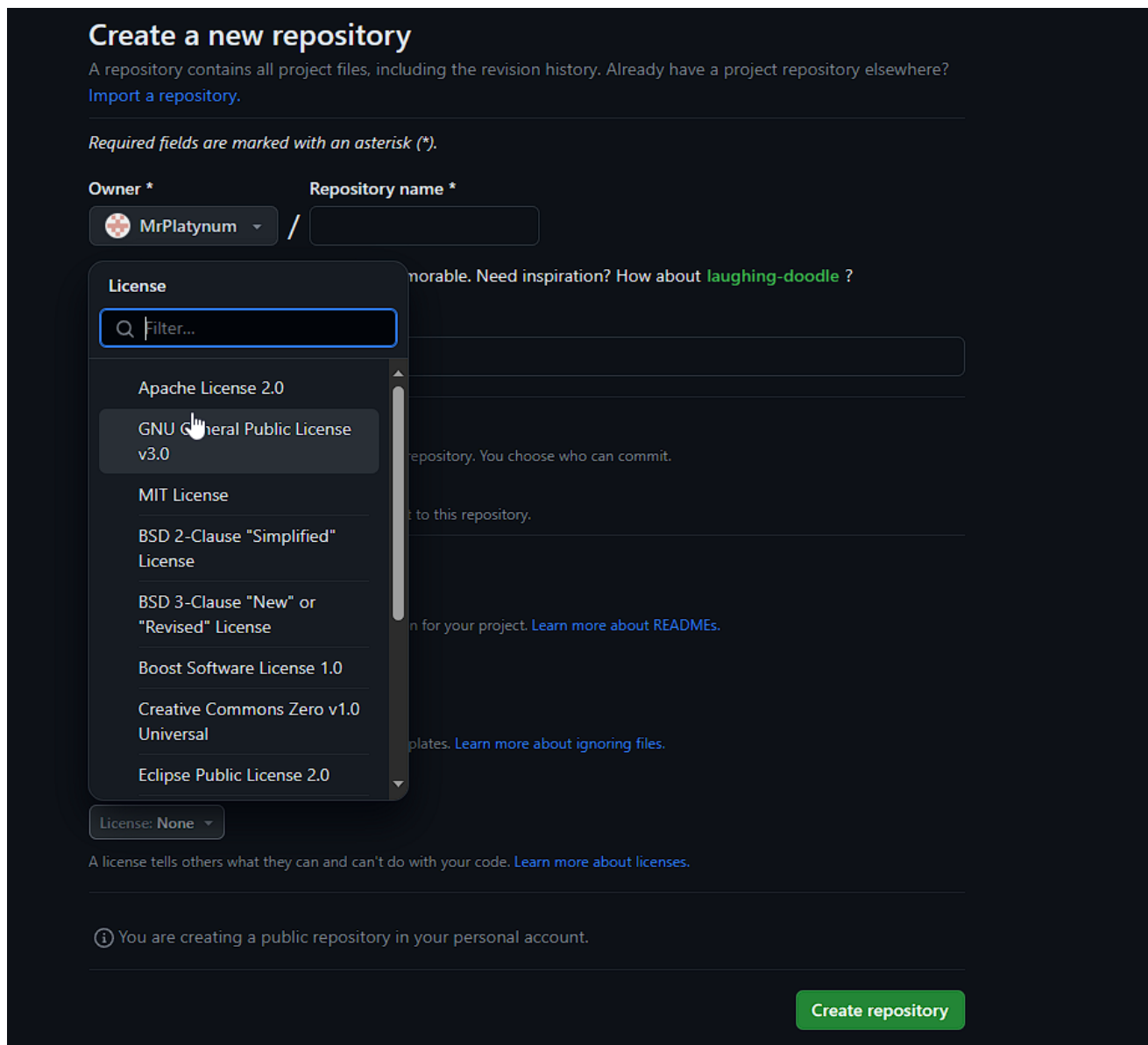


図4 – ライセンスの選択

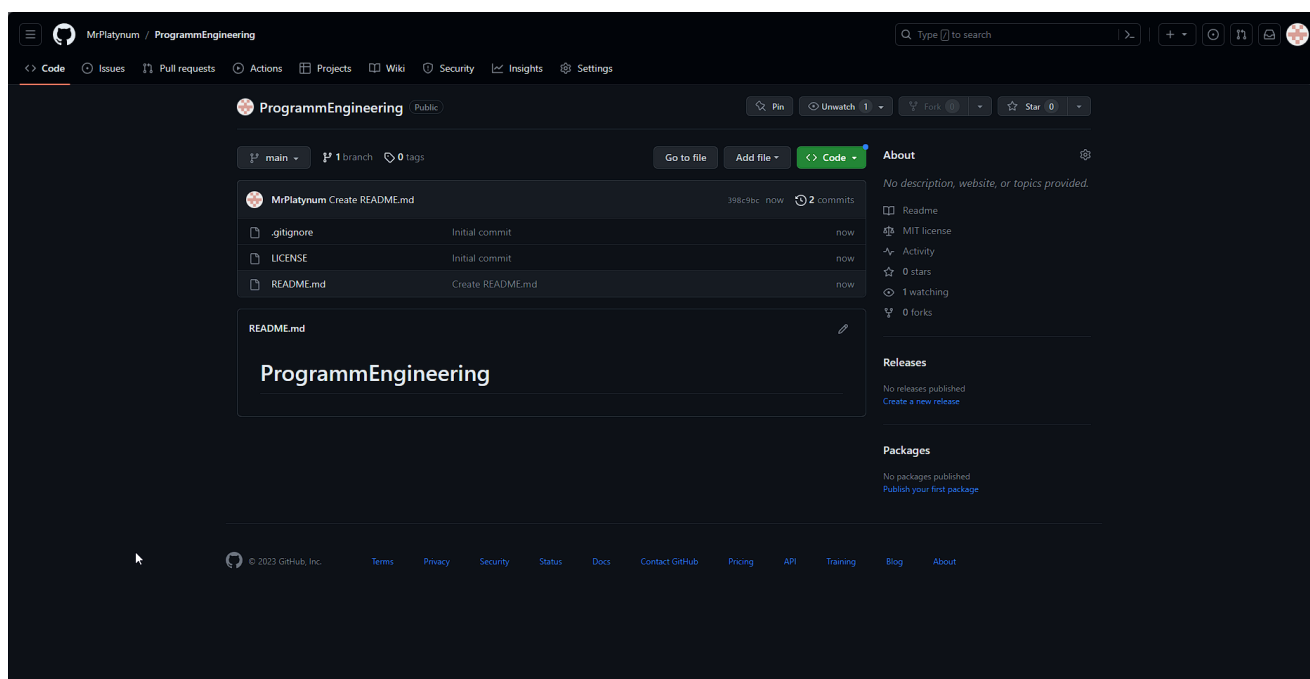
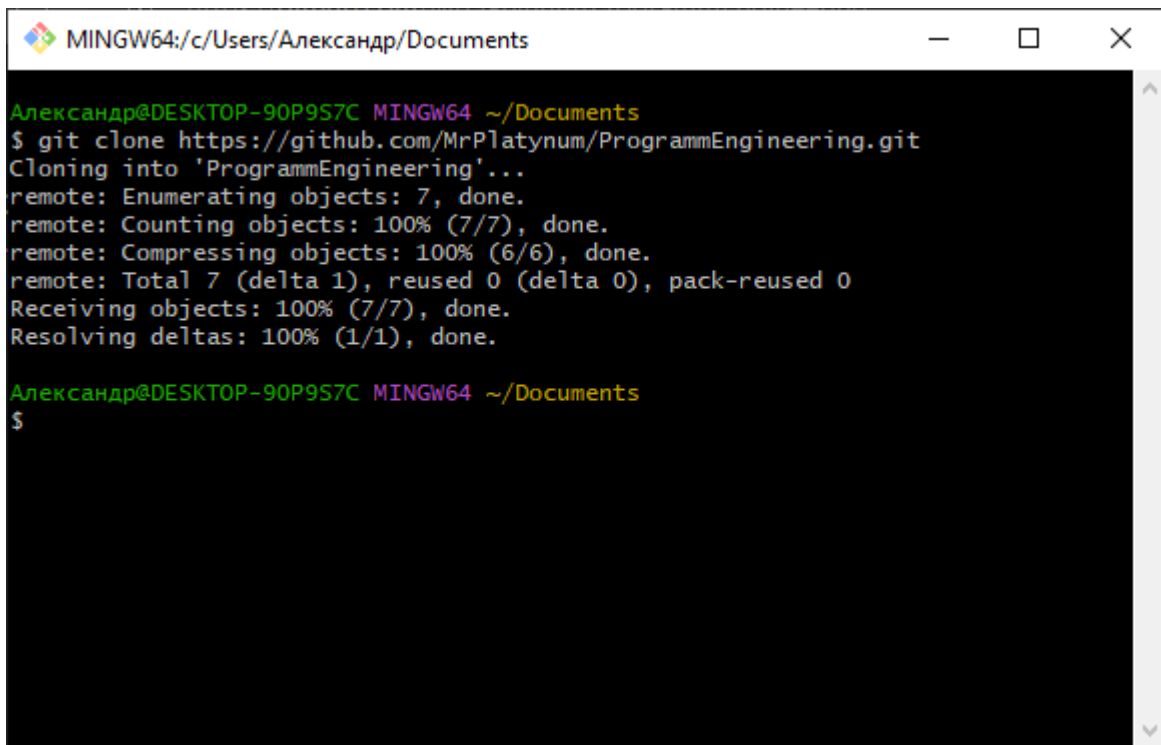


図5 – 作成されたりポジトリ

1. 作成したリポジトリをローカルコンピュータにクローンします。



```
MINGW64:/c/Users/Александр/Documents
Александр@DESKTOP-90P9S7C MINGW64 ~/Documents
$ git clone https://github.com/MrPlatynum/ProgrammEngineering.git
Cloning into 'ProgrammEngineering'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (7/7), done.
Resolving deltas: 100% (1/1), done.

Александр@DESKTOP-90P9S7C MINGW64 ~/Documents
$
```

図6 – git clone コマンドを使用してリポジトリをクローンする

2. .gitignore の内容を表示します。



```
.gitignore
# Byte-compiled / optimized / DLL files
__pycache__ /
*.py[co]
*.pyc
*.pyc.class

# C extensions
*.so

# Distribution / packaging
Python
build/
develop-eggs/
dist/
downloads/
eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
*.installed.cfg
*.egg
MANIFEST

# PyInstaller
# Usually these files are written by a python script from a template
# before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

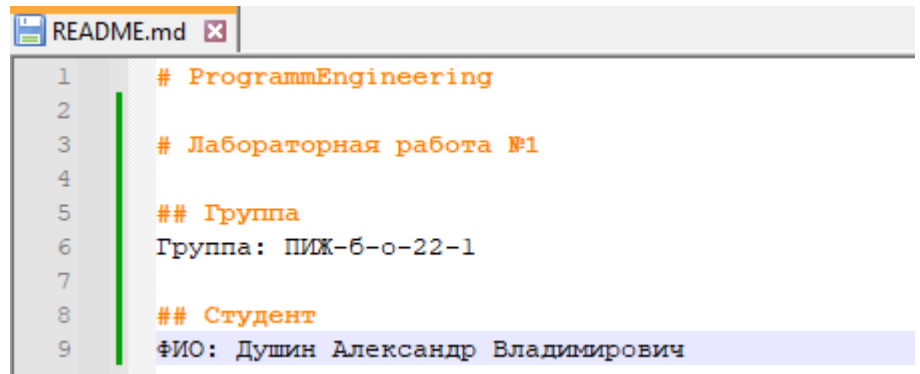
# Unit test / coverage reports
htmlcov/
.tox/
nosetests/
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.py,cover
*.hypothesis/
pytest_cache/
cov/

# Translations
*.mo
*.pot
```

図7 – .gitignore ファイルの内容

3. README.md ファイルに、ラボを実行している学生のグループとフルネームに関

する情報を追加します。

A screenshot of a code editor window titled 'README.md'. The editor shows a list of 9 lines of code. Line 1: # ProgrammEngineering. Line 2: (empty). Line 3: # Лабораторная работа №1. Line 4: (empty). Line 5: ## Группа. Line 6: Группа: ПИЖ-6-о-22-1. Line 7: (empty). Line 8: ## Студент. Line 9: ФИО: Душин Александр Владимирович. The text is in a monospaced font with syntax highlighting: '#' is orange, '##' is orange, and the text after the markers is blue. The line numbers 1 through 9 are in a light gray margin on the left.

```
1 # ProgrammEngineering
2
3 # Лабораторная работа №1
4
5 ## Группа
6 Группа: ПИЖ-6-о-22-1
7
8 ## Студент
9 ФИО: Душин Александр Владимирович
```

図8 – README.md ファイルへの情報の追加

4. 選択したプログラミング言語で小さなプログラムを書きます。ローカルリポジトリで変更をコミットします。

MINGW64:/c/Users/Александр/Documents/ProgrammEngineering

commit 982be340feaa9e99b298efb77e4ea8c49cdc1ebd (HEAD -> main)
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:15:24 2023 +0300

Добавление комментариев

commit d58423e629ee2b053fe7966a4514f436aaebcef
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:13:11 2023 +0300

Проверка нечётности числа

commit 963b3f50faf631f523463103fd68da367582d2f1
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:12:45 2023 +0300

Проверка чётности числа

commit 15c1ee84ba4c68340a81b6826fa7f1595a1a1165
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:11:10 2023 +0300

Создание функции ввода числа

commit 6670d83a98c90732cea9b256f96586a3a9856a24
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:10:13 2023 +0300

Создание дополнения условия функции

commit 4371cf4936db4345b3d2205817c4e29a2302d183
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:09:10 2023 +0300

Создание условия функции

commit d621ef9437d7229f8ae59170ad2b3506cf788445
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:08:03 2023 +0300

Создание функции в файле

commit c67a1d2853edc7ee404e3146fa59a5793d6c3325
Author: MrPlatynum <dushinsasha4@gmail.com>
Date: Tue Sep 12 22:06:16 2023 +0300

Создание файла python

図9 – コミットの履歴

```
Hello.py  ▢ ×
def check_even(number):
    if number % 2 == 0:
        return True
    else:
        return False

# Получение числа от пользователя
number = int(user_input)

# Проверка четности числа и вывод результата
if check_even(number):
    print("Число", number, "является четным.")
else:
    print("Число", number, "является нечетным.")
```

図10 – プログラムのコード

4. `README.md` ファイルで行った変更をコミットします。

```
README.md  ▢ ×
1  # ProgrammEngineering
2
3  # Лабораторная работа №1
4
5  ## Группа
6  Группа: ПИЖ-б-о-22-1
7
8  ## Студент
9  ФИО: Душин Александр Владимирович
10
11 ##Описание
12 Цель данной работы исследовать базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.
13
14 Программа просит пользователя ввести число и выводит чётность этого числа.
```

図11 – 最終的な README.md ファイル

Ответы на контрольные вопросы:

1. とは何か、そしてその目的は何ですか？

ソースコード管理システム (SCV) は、1つまたは複数のファイルでの変更を記録し、これらのファイルの特定の古いバージョンに後で戻ることができるシステムです。

2. ローカルおよび中央集権型のバージョン管理システムの欠点は何ですか？

多くの人々は、バージョン管理の方法としてファイルを別のディレクトリにコピーすることを採用しています（場合によっては、時間刻みのディレクトリを使用することもあります）。このアプローチはその単純さから非常に一般的ですが、誤りの発生に非常に脆弱です。どのディレクトリにいるのかを簡単に忘れ、誤って誤ったファイルを変更したり、意図しないファイルをコピーしたりすることがあります。

最も明らかなデメリットは、中央集権的なサーバーによる単一の障害点です。このサーバーが1時間でもダウンした場合、その間バージョン管理を使用して作業することができず、他の開発者とこれらの変更を共有することもできません。中央データベースを格納しているハードディスクが壊れ、適切なバックアップがない場合、プロジェクトの履歴全体を失うリスクがあります。ローカルのSCVも同じ問題に苦しんでおり、プロジェクトの履歴全体が1か所に格納されているため、すべてを失う可能性があります。

3. はどのバージョン管理システムに属していますか？

Gitは分散型のバージョン管理システムです。

4. Gitと他のバージョン管理システムとの概念的な違いは何ですか？

Gitと他のバージョン管理システムとの主要な違いは、データの取り扱い方です。

概念的には、ほとんどの他のシステムはファイルの変更のリストとしてデータを保存します。これらのシステムはファイルと各ファイルの変更を時間軸で保存します（通常、差分に基づくバージョン管理と呼ばれます）。Gitはこのような方法でデータを保存せず、データの取り扱いにおいてよりファイルシステムのスナップショットのセットに似ています。つまり、コミット（プロジェクトの状態をGitに保存すること）のたびに、Gitはその瞬間の各ファイルの外観を記録し、そのスナップショットへのリンクを保存します。

5. Gitで格納されたデータの整合性はどのように確保されていますか？

Gitはデータの整合性をハッシュ制御サムによって確保しています。Git内の各オブジェクト、ファイル、コミット、またはブランチは、固有のハッシュを持っています。オブジェクトがリポジトリに保存されると、Gitはそのハッシュを計算し、オブジェクトをそのハッシュの下に保存します。後でオブジェクトを読み込むと、Gitは再びハッシュを計算し、保存されたハッシュと比較します。ハッシュが一致する場合、データは変更されず正しく保存されていることを示します。ハッシュが一致しない場合、これはデータの変更またはストレージシステムのエラーを示します。さらに、Gitはデータの整合性を確保するためにオブジェクトツリーを使用します。オブジェクトツリーは、各コミットが前のコミット、ファイル、および他のオブジェクトを指すグラフィカルな構造です。ツリー内のすべてのコミットとオブジェクトは、それらのハッシュを使用して保存され、データの整合性を確保します。

6. Gitでは、ファイルはどのような状態になることができますか？これらの状態はどのように関連していますか？

Gitにはファイルが存在する可能性のある3つの主要な状態があります：コミットさ

れた (committed) 状態、変更された (modified) 状態、およびステージングされた (staged) 状態です。

特定のファイルのバージョンがGitディレクトリに存在する場合、そのバージョンは「コミットされた」と見なされます。

ファイルのバージョンが変更され、インデックスに追加されている場合、それは「ステージングされた」状態です。そして、ファイルが最後にリポジトリからチェックアウトされてから変更されたが、インデックスに追加されていない場合、そのファイルは「変更された」と見なされます。

7. GitHubのユーザープロフィールとは何ですか？

プロフィールは、GitHub上のあなたの公開ページで、ソーシャルメディアのようなものです。プログラマーとしての仕事を探す際、雇用主はあなたのGitHubプロフィールを見て、あなたを雇うかどうかを決定する際に考慮することがあります。

8. GitHubのリポジトリにはどのような種類がありますか？

ローカル - 1台のコンピュータにあり、1人しか利用できません。

中央集権 - サーバーにあり、複数のプログラマーが同時にアクセスできます。

分散 - クラウドアクセスを備えた最も便利な選択肢です。

9. GitHubでの作業モデルの主要なステップを指摘してください。

最初にGitHubの領域を検討します。2つのリポジトリがあります。upstreamはプロジェクトの元のリポジトリで、コピーしたものです。originはGitHub上のあなたのフォーク（コピー）で、完全なアクセス権があります。あなたのコピーから元のプロジェクトのリポジトリ

リに変更を持って行くには、プルリクエストを作成する必要があります。小規模な変更をあなたのコピー（フォーク）に加えたい場合、GitHubのウェブインターフェースを使用できます。ただし、プログラムの開発中は頻繁にローカルで実行およびデバッグする必要があるため、この方法は便利ではありません。標準的な方法は、リモートリポジトリのローカルクローンを作成し、ローカルで作業し、定期的にリモートリポジトリに変更を反映させることです。

10. Gitをインストールした後、最初の設定はどのように行われますか？

Gitが正常にインストールされたことを確認するために、次のコマンドをターミナルに入力して、Gitの現在のバージョンを表示します：

```
git version
```

それが機能した場合、Gitの設定にあなたの名前、姓、およびGitHubアカウントに関連付けられた電子メールアドレスを追加しましょう：

```
git config --global user.name git config --global user.email
```

11. GitHubでリポジトリを作成する手順を説明してください。

右上隅に、アバターの横にプラス記号のボタンがあり、それをクリックすると新しいリポジトリの作成ページに移動します。

その結果、リポジトリの作成ページに移動します。このページで最も重要な項目は次のとおりです：

- リポジトリ名。これは一意である必要はありませんが、github全体では一意でなくても、あなたのアカウントに関連しています。ただし、あなたが作成したリポジトリ内では一意である必要があります。

- 説明（Description）。空白のままにすることもできます。
- Public/private。オープン（Public）を選択し、「Initialize this repository with a README」のチェックボックスをオフにします（READMEにはプロジェクトに関する基本情報が含まれ、プロジェクトとの作業方法が説明されます）。
- .gitignoreおよびLICENSEは今選択する必要はありません。これらのフィールドを入力した後、Create repositoryボタンをクリックします。

12. GitHubのリポジトリを作成する際にサポートされているライセンスの種類は何か？

GitHubは、MIT License、GNU GPL、Apache License、Creative Commons Licenses、Unlicenseなど、さまざまなタイプのライセンスをサポートしています。

13. GitHubリポジトリのクローン作成方法と、リポジトリのクローンが必要な理由は何ですか？

ローカルコンピュータにリポジトリをクローンするには、リポジトリページで「Clone」または「Code」ボタンを見つけてクリックし、クローンするリポジトリのアドレスを表示します。

コマンドラインまたはターミナルを開いて、リポジトリをコピーしたいディレクトリに移動します。その後、`git clone`と入力し、アドレスを入力します：

```
git clone https://github.com/kushedow/flask-html
```

14. ローカルGitリポジトリの状態を確認する方法は？

ローカルリポジトリの状態を確認するには、次のコマンドを入力してください：

```
git status
```

15. 以下の操作を実行した後、Gitのローカルリポジトリの状態はどのように変化

しますか:

"git add"コマンドは、作業ディレクトリ内の変更をステージングエリアに追加します。これにより、Gitに対して指定したファイルの変更を次のコミットに含めることを伝えます。

"git commit"コマンドは、ステージングエリア内の変更のスナップショットをリポジトリのコミット履歴に保存します。

重要なことは、"git commit"は変更をローカルリポジトリにのみ追加します。変更をGitHubのリモートリポジトリにプッシュしたい場合、"git push"を使用する必要があります。最初の回では、ローカルブランチがリモートリポジトリに存在しないため、次のコマンドを使用してローカルブランチをリモートリポジトリにプッシュする必要があります：

```
git push --set-upstream origin edit-readme
```

次回からは、簡単にプッシュできるようになります：

```
git push
```

16. GitHubにリポジトリがあり、このリポジトリを使用していくつかのプロジェクトで作業できる2つの作業用コンピュータがあります。GitHubリポジトリに関連付けられている両方のローカルリポジトリが同期状態になるコマンドのシーケンスを記述します。注:説明はgit cloneコマンドで始める必要があります。

それぞれのコンピュータでターミナルまたはコマンドラインを開きます。

各コンピュータで、ローカルリポジトリのコピーを保存したいディレクトリに移動します。

以下のようにgit cloneコマンドを使用してGitHubからリポジトリをクローンします：

`git clone <リポジトリのURL>`。例：`git clone https://github.com/username/repo.git`。

その後、一方のコンピュータでリポジトリのディレクトリに移動し、`cd repo`コマンドを使用します。

一方のコンピュータで、`git remote add`コマンドを使用して、別のコンピュータへのリンクを最初のコンピュータのリモートリポジトリとして追加します。コマンドは以下のようになります：`git remote add <別のコンピュータの名前> <別のコンピュータのURL>`。
例：`git remote add second_computer <別のコンピュータのURL>`。

両方のコンピュータで、`git pull`コマンドを使用してローカルリポジトリをGitHubのリモートリポジトリと同期させます。コマンドは以下のようになります：`git pull <リモートリポジトリの名前> <ブランチ名>`。例：`git pull origin master`。必要に応じて、両方のコンピュータでこのコマンドを実行します。

これで、GitHubのリポジトリに関連付けられた2つのローカルリポジトリが同期されるはずです。プロジェクトをどちらかのコンピュータで作業し、変更をGitHubにアップロードするには、`git push`コマンドを使用します。例：`git push origin master`。もう一方のコンピュータは、`git pull`コマンドを使用してこれらの変更を取得できます。例：`git pull origin master`。

17. Gitで動作するサービスはGitHubだけではありません。他にどのようなサービスが知られていますか？これらのサービスの1つをGitHubと比較分析します。

GitHubとGitLabは、Gitを使用したプロジェクト管理およびソースコードホスティングのための2つの主要なプラットフォームです。GitHubは、広く使用されており、大規模な

コミュニティ、多くのツールとサービスの統合、オープンソースプロジェクト向けの無料プランなどを提供しています。GitLabは、CI/CD（継続的インテグレーション/継続的デリバリー）の強力な統合、オンプレミスでのセルフホステッドオプション、より機能豊富な無料プランなどを提供しており、特に開発者向けに非常に魅力的です。

どちらを選ぶかは、プロジェクトのニーズや個人の好みに依存します。GitHubは特にオープンソースプロジェクトに適しており、GitLabはCI/CDと自己ホスティングが必要な場合に有用です。プライベートリポジトリに対しては、GitLabの無料プランの方が機能が豊富です。選択はプロジェクトの要件に合わせて行うべきです。

18. コマンドラインインターフェイスだけではなく、Gitを操作する最も便利な方法からはほど遠いものです。Gitを操作するためのグラフィカルユーザーインターフェイスでどのようなソフトウェアツールを知っていますか？これらのソフトウェアツールのいずれかを使用して、実験室での作業で説明されているGit操作をどのように実装するかを示します。

GitHub Desktop、GitKraken、SourceTree、およびTortoiseGitなど、多くのGUIツールがGitをサポートしています。これらのツールを使用すると、リポジトリのクローン、コミットの作成、および変更のリモートリポジトリへのプッシュなど、Git操作を簡単に行うことができます。