

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования «СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ  
УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе №1.1

Дисциплина: «Основы кроссплатформенного программирования»

Тема: «Исследование возможностей повторного использования кода в  
структурном программировании»

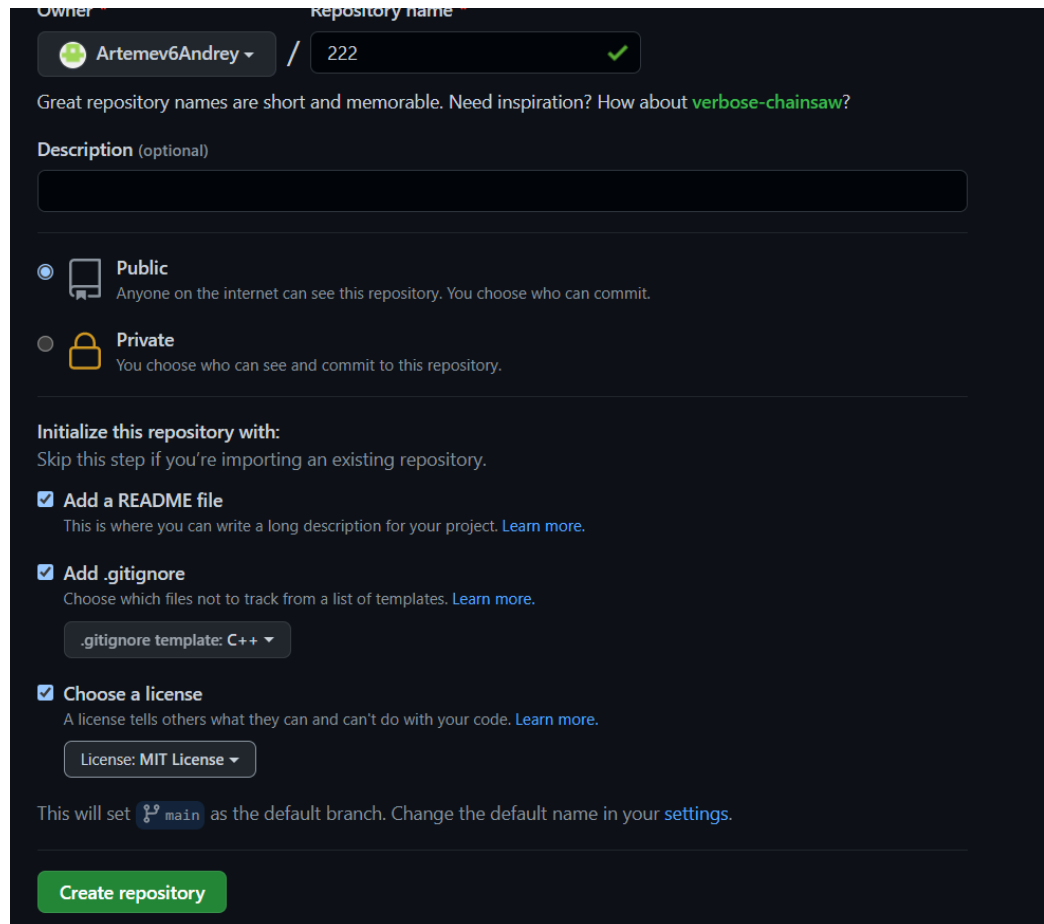
Выполнил: студент 1 курса,  
группы ИВТ-б-о-21-1  
Артемьев Андрей Витальевич

Ставрополь 2022

Цель работы: исследовать базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub

## Выполнение работы:

### 1. Создал и настроил репозиторий



Owner: Artemev6Andrey / Repository name: 222

Great repository names are short and memorable. Need inspiration? How about **verbose-chainsaw**?

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:  
Skip this step if you're importing an existing repository.

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

☒ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)  
.gitignore template: C++

☒ **Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)  
License: MIT License

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

**Create repository**

Рисунок 2. Создание и настройка репозитория

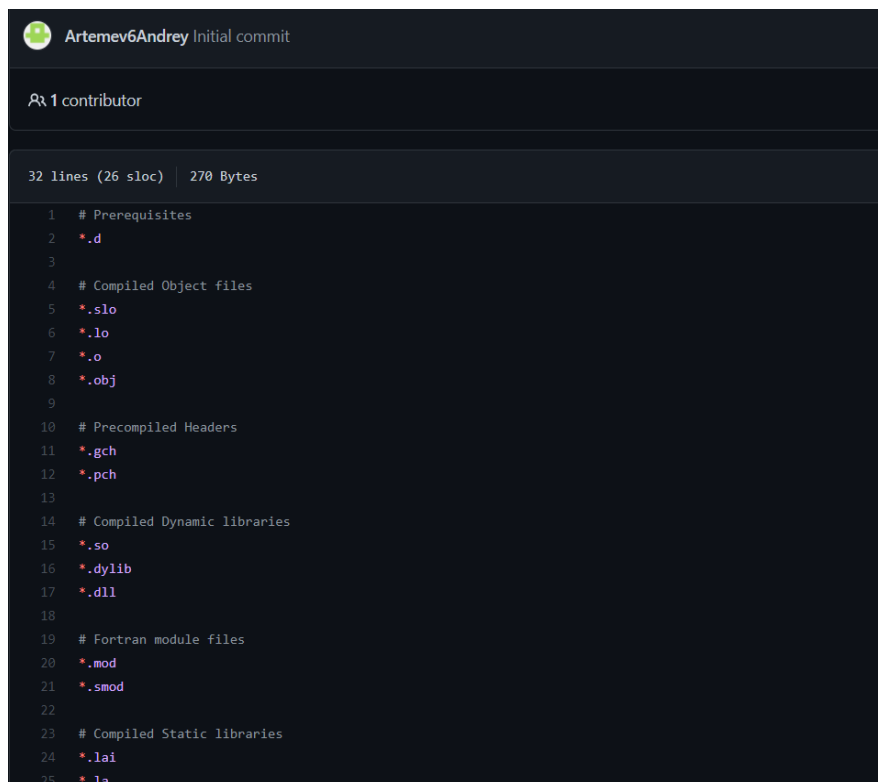
### 2. Клонировал репозиторий на компьютер

```
C:\Users\aa715\Desktop\222>git clone https://github.com/Artemev6Andrey/222.git
Cloning into '222'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 11 (delta 3), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (11/11), done.
Resolving deltas: 100% (3/3), done.
C:\Users\aa715\Desktop\222>
```

Рисунок 3. Клонирование репозитория

### 3. Дополнил файл .gitignore необходимыми правилами для выбранного

языка программирования и интегрированной среды разработки.

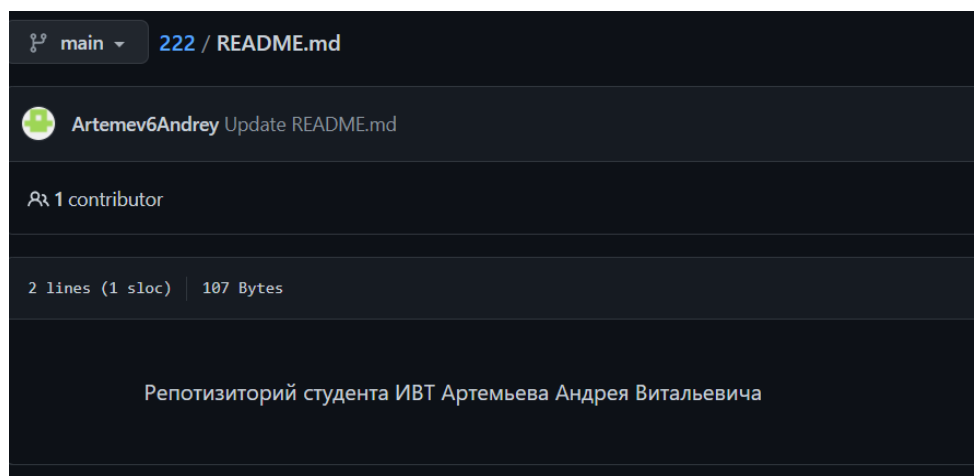


The screenshot shows a GitHub commit interface for a repository named 'Artemev6Andrey'. The commit is titled 'Initial commit' and is attributed to 'Artemev6Andrey'. It shows a single contributor. The commit message is '32 lines (26 sloc) | 270 Bytes'. The file being committed is '.gitignore', which contains the following content:

```
1 # Prerequisites
2 *.d
3
4 # Compiled Object files
5 *.slo
6 *.lo
7 *.o
8 *.obj
9
10 # Precompiled Headers
11 *.gch
12 *.pch
13
14 # Compiled Dynamic libraries
15 *.so
16 *.dylib
17 *.dll
18
19 # Fortran module files
20 *.mod
21 *.smod
22
23 # Compiled Static libraries
24 *.lai
25 *.la
```

Рисунок 4. Внесение изменений в .gitignore

4. Изменение файла README.



The screenshot shows a GitHub commit interface for a repository named 'Artemev6Andrey'. The commit is titled 'Update README.md' and is attributed to 'Artemev6Andrey'. It shows a single contributor. The commit message is '2 lines (1 sloc) | 107 Bytes'. The file being committed is 'README.md', which contains the following content:

```
Репозиторий студента ИВТ Артемьева Андрея Витальевича
```

Рисунок 5. Изменение файла README

5. Написал программу.

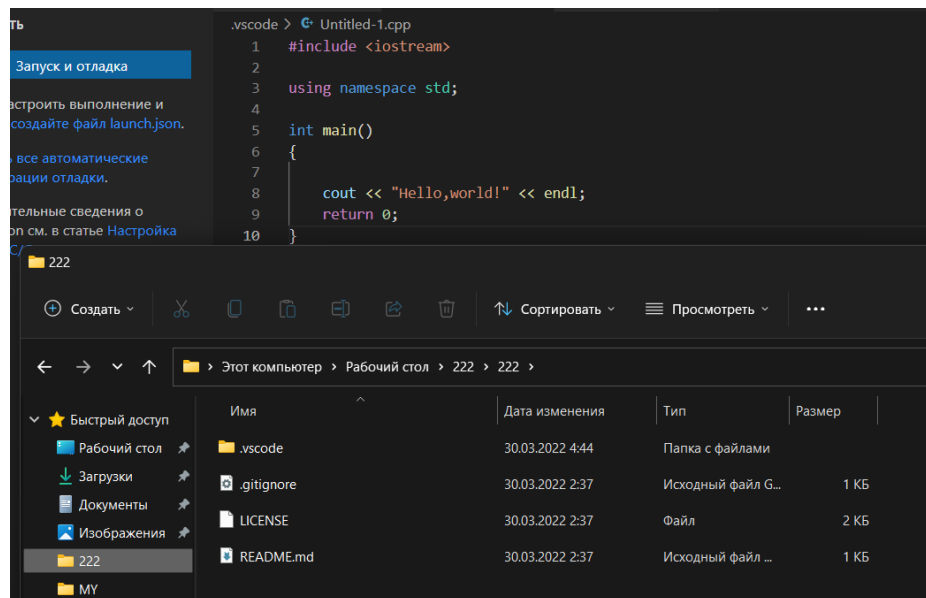


Рисунок 6. Код программы

## 6. Работа с консолью

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "Programm Hello world"
[main 42b7ba1] Programm Hello world
3 files changed, 44 insertions(+)
create mode 100644 .vscode/Untitled-1.cpp
create mode 100644 .vscode/launch.json
create mode 100644 .vscode/tasks.json

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git pull
Already up to date.

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.21 KiB | 1.21 MiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Artemev6Andrey/222.git
6276d66..42b7ba1 main -> main

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$

```

Рисунок 7. Работа консоли

## 7. Отследил изменения на удаленном репозитории

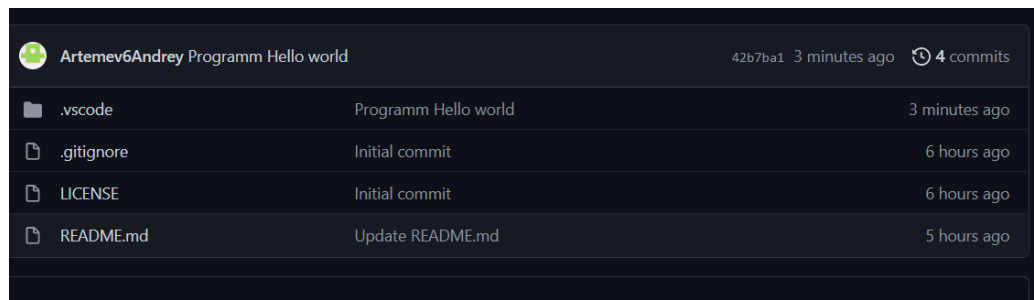


Рисунок 8. Изменения на удаленном репозитории

8. Зафиксировал изменения при написании программы на локальном репозитории и ввел команду push в консоли.

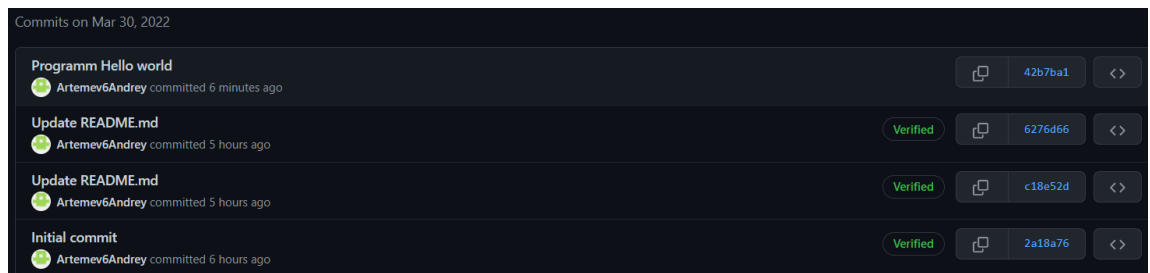


Рисунок 9. Список коммитов в github-e

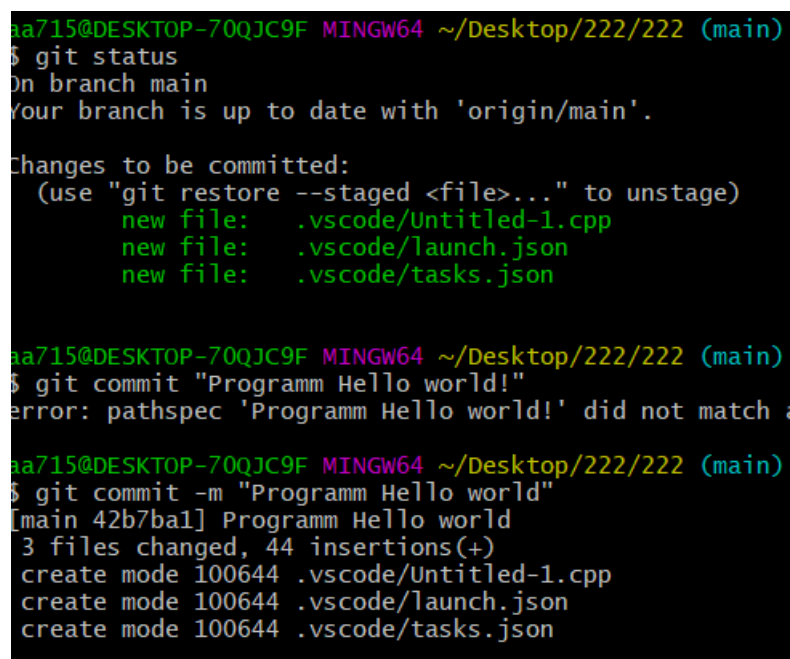


Рисунок 10. Изменение программы и 1-ый коммит

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      cout << "Hello,world!" << endl;
9      cout<<1+1;
10     return 0;
11 }

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "2"
[main 865a486] 2
1 file changed, 1 insertion(+)

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$

```

Рисунок 11. Изменение программы и 2-ой коммит

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      cout << "Hello,world!" << endl;
9      cout<<1+1;
10     cout <<"Hello world!\n";
11     return 0;
12 }

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "3"
[main 6cff68d] 3
1 file changed, 1 insertion(+)

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$

```

Рисунок 12. Изменение программы и 3-ий коммит

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      cout << "Hello,world!" << endl;
9      cout<<1+1;
10     cout <<"Hello world!\n";
11     cout<<"World,23!";
12     return 0;
13 }

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "4"
[main 8d8b07b] 4
1 file changed, 1 insertion(+)
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$

```

Рисунок 13. Изменение программы и 4-ый коммит

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      cout << "Hello,world!" << endl;
9      cout<<1+1;
10     cout <<"Hello world!\n";
11     cout<<"\nWorld,23!";
12     int a, b;
13     return 0;
14 }

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "5"
[main 25b3ade] 5
1 file changed, 2 insertions(+), 1 deletion(-)
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$

```

Рисунок 14. Изменение программы и 5-ый коммит

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7
8      cout << "Hello,world!" << endl;
9      cout <<1+1;
10     cout <<"Hello world!\n";
11     cout <<"\nWorld,23!";
12     int a, b;
13     return 0;
14     cout <<2+3;
15 }

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "6"
[main eed115e] 6
1 file changed, 3 insertions(+), 2 deletions(-)

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ |

```

Рисунок 15. Изменение программы и 6-ой коммит

```

#include <iostream>

using namespace std;

int main()
{

    cout << "Hello,world!" << endl;
    cout <<1+1;
    cout <<"Hello world!\n";
    cout <<"\nWorld,23!";
    int a, b;
    cout <<2+3;
    cout <<5+5;
    return 0;
}

```

```

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git add .

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git commit -m "7"
[main 697487b] 7
1 file changed, 2 insertions(+), 1 deletion(-)

aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)

```

Рисунок 16. Изменение программы и 7-ой коммит



Сделал push репозитория на удаленный репозиторий:



```
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 8 threads
Compressing objects: 100% (24/24), done.
Writing objects: 100% (24/24), 2.24 KiB | 2.24 MiB/s, done.
Total 24 (delta 11), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (11/11), completed with 1 local object.
To https://github.com/Artemev6Andrey/222.git
 42b7ba1..697487b  main -> main
aa715@DESKTOP-70QJC9F MINGW64 ~/Desktop/222/222 (main)
$ |
```

Рисунок 17. Ввод команды push в консоли

## Ответы на вопросы:

### 1. Что такое СКВ и каково ее назначение?

Система контроля версий (СКВ) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

### 2. В чем недостатки локальных и централизованных СКВ?

Локальные СКВ: многие в качестве метода контроля версий применяют копирование файлов в отдельную директорию. Такой подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Централизованные СКВ: единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками.

### 3. К какой СКВ относится Git?

Git относится к распределённым системам контроля версий.

### 4. В чем концептуальное отличие Git от других СКВ?

Основное отличие Git от любой другой СКВ (включая Subversion и её собратьев) — это подход к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы (CVS, Subversion, Perforce, Bazaar и т. д.) представляют хранимую информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени (обычно это называют контролем версий, основанным на различиях). Git не хранит и не обрабатывает данные таким способом. Вместо этого, подход Git к хранению данных больше похож на набор снимков миниатюрной файловой системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, поток снимков

## **5. Как обеспечивается целостность хранимых данных в Git?**

В Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме. Это значит, что невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом. Данная функциональность встроена в Git на низком уровне и является неотъемлемой частью его основы. В итоге информация не теряется во время её передачи и файл не повредится без ведома Git.

## **6. В каких состояниях могут находиться файлы в Git?**

Как связаны эти состояния? У Git есть три основных состояния, в которых могут находиться ваши файлы: зафиксированное (committed), изменённое (modified) и подготовленное (staged).

Зафиксированный значит, что файл уже сохранён в вашей локальной базе.

К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы.

Подготовленные файлы — это изменённые файлы, отмеченные для включения в следующий коммит.

## **7. Что такое профиль пользователя в GitHub?**

Профиль - это ваша публичная страница на GitHub, как и в социальных сетях. Когда вы ищете работу в качестве программиста, работодатели могут посмотреть ваш профиль GitHub и принять его во внимание, когда будут решать, брать вас на работу или нет.

## **8. Какие бывают репозитории в GitHub?**

Локальный репозиторий — это подкаталог `.git`, создаётся (в пустом виде) командой `git init` и (в непустом виде с немедленным копированием содержимого родительского удалённого репозитория и простановкой ссылки на родителя) командой `git clone`. Практически все обычные операции с системой контроля версий, такие, как коммит и слияние, производятся только с локальным репозиторием.

Удалённый доступ к репозиториям Git обеспечивается `git-daemon`, SSH или HTTP-сервером. TCP-сервис `git-daemon` входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Удалённый репозиторий можно только синхронизировать с локальным как «вверх» (`push`), так и «вниз» (`pull`).

## **9. Укажите основные этапы модели работы с GitHub.**

- 1) регистрация;
- 2) создание репозитория;
- 3) клонирование репозитория.

## **10. Как осуществляется первоначальная настройка Git после установки?**

- 1) убедимся, что Git установлен используя команду: `git version`;
- 2) перейдём в папку с локальным репозиторием, используя команду: `cd /d`;

3) свяжем локальный репозиторий и удалённый командами: `git config --global user.name` `git config --global user.email`.

## **11. Опишите этапы создания репозитория в GitHub.**

В правом верхнем углу, рядом с аватаром есть кнопка с плюсиком, нажимая которую мы переходим к созданию нового репозитория.

В результате будет выполнен переход на страницу создания репозитория. Наиболее важными на ней являются следующие поля:

Имя репозитория. Оно может быть любое, необязательно уникальное во всем github, потому что привязано к вашему аккаунту, но уникальное в рамках тех репозиторий, которые вы создавали. Описание (Description). Можно оставить пустым. Public/private. Выбираем открытый (Public), НЕ ставим галочку “Initialize this repository with a README” ( В README потом будет лежать какая-то основная информация, что же такое ваш проект и как с ним работать). .gitignore и LICENSE можно сейчас не выбирать.

После заполнения этих полей нажимаем кнопку Create repository. Отлично, ваш репозиторий готов!

## **12. Какие типы лицензий поддерживаются GitHub при создании репозитория?**

Microsoft Reciprocal License, The Code Project Open License (CPO), The Common Development and Distribution License (CDDL), The Microsoft Public License (Ms-PL), The Mozilla Public License 1.1 (MPL 1.1), The Common Public License Version 1.0 (CPL), The Eclipse Public License 1.0, The MIT License, The BSD License, The Apache License, Version 2.0, The Creative Commons Attribution-ShareAlike 2.5 License, The zlib/libpng License, A Public Domain dedication, The Creative Commons Attribution 3.0 Unported License, The Creative Commons Attribution-Share Alike 3.0 Unported License, The Creative Commons Attribution-NoDerivatives 3.0 Unported, The GNU Lesser General Public License (LGPLv3), The GNU General Public License (GPLv3).

## **13. Как осуществляется клонирование репозитория GitHub? Зачем нужно клонировать репозиторий?**

После создания репозитория его необходимо клонировать на ваш компьютер. Для этого на странице репозитория необходимо найти кнопку Clone или Code и щелкнуть по ней, чтобы отобразить адрес репозитория для клонирования.

#### **14. Как проверить состояние локального репозитория Git?**

Локальный репозиторий включает в себя все те же файлы, ветки и историю коммитов, как и удаленный репозиторий. Введите эту команду, чтобы проверить состояние вашего репозитория: `git status`.

**15. Как изменяется состояние локального репозитория Git после выполнения следующих операций: добавления/изменения файла в локальный репозиторий Git; добавления нового/измененного файла под версионный контроль с помощью команды `git add` ; фиксации (коммита) изменений с помощью команды `git commit` и отправки изменений на сервер с помощью команды `git push` ?**

Коммит добавляет изменения только в ваш локальный репозиторий. Если вы хотите распространить их в исходный репозиторий на GitHub, вам нужно использовать `push`.

**16. У Вас имеется репозиторий на GitHub и два рабочих компьютера, с помощью которых Вы можете осуществлять работу над некоторым проектом с использованием этого репозитория. Опишите последовательность команд, с помощью которых оба локальных репозитория, связанных с репозиторием GitHub будут находиться в синхронизированном состоянии.**

Примечание: описание необходимо начать с команды `git clone`.

1) Клонировать репозиторий на каждый из компьютеров, используя команду `git clone` и ссылку.

2) Для синхронизации изменений используем команду `git pull`.

**17. GitHub является не единственным сервисом, работающим с Git. Какие сервисы еще Вам известны? Приведите сравнительный анализ одного из таких сервисов с GitHub.**

GitLab — альтернатива GitHub номер один. GitLab предоставляет не только веб-сервис для совместной работы, но и программное обеспечение с открытым исходным кодом.

SourceForge — ещё одна крупная альтернатива GitHub, сконцентрировавшаяся на Open Source. Многие дистрибутивы и приложения Linux обитают на SourceForge.

Launchpad — платформа для совместной работы над программным обеспечением от Canonical, компании-разработчика Ubuntu. На ней размещены PPA-репозитории Ubuntu, откуда пользователи загружают приложения и обновления.

**18. Интерфейс командной строки является не единственным и далеко не самым удобным способом работы с Git. Какие Вам известны программные средства с графическим интерфейсом пользователя для работы с Git? Приведите как реализуются описанные в лабораторной работе операции Git с помощью одного из таких программных средств.**

1) GitHub Desktop это бесплатное приложение с открытым исходным кодом, разработанное GitHub. С его помощью можно взаимодействовать с GitHub, а также с другими платформами (включая GitLab).

2) Fork это весьма продвинутый GUI-клиент для macOS и Windows (с бесплатным пробным периодом). В фокусе этого инструмента скорость, дружелюбность к пользователю и эффективность. К особенностям Fork можно отнести красивый вид, кнопки быстрого доступа, встроенную систему разрешения конфликтов слияния, менеджер репозитория, уведомления GitHub.

3) Sourcetree это бесплатный GUI Git для macOS и Windows. Его применение упрощает работу с контролем версий и позволяет сфокусироваться на действительно важных задачах.

4) martGit это Git-клиент для Mac, Linux и Windows. Имеет богатый функционал. В арсенале SmartGit вы найдете CLI для Git, графическое

отображение слияний и истории коммитов, SSH-клиент, Git-Flow, программу для разрешения конфликтов слияния.

**Вывод:** в результате выполнения работы я исследовала базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub.