

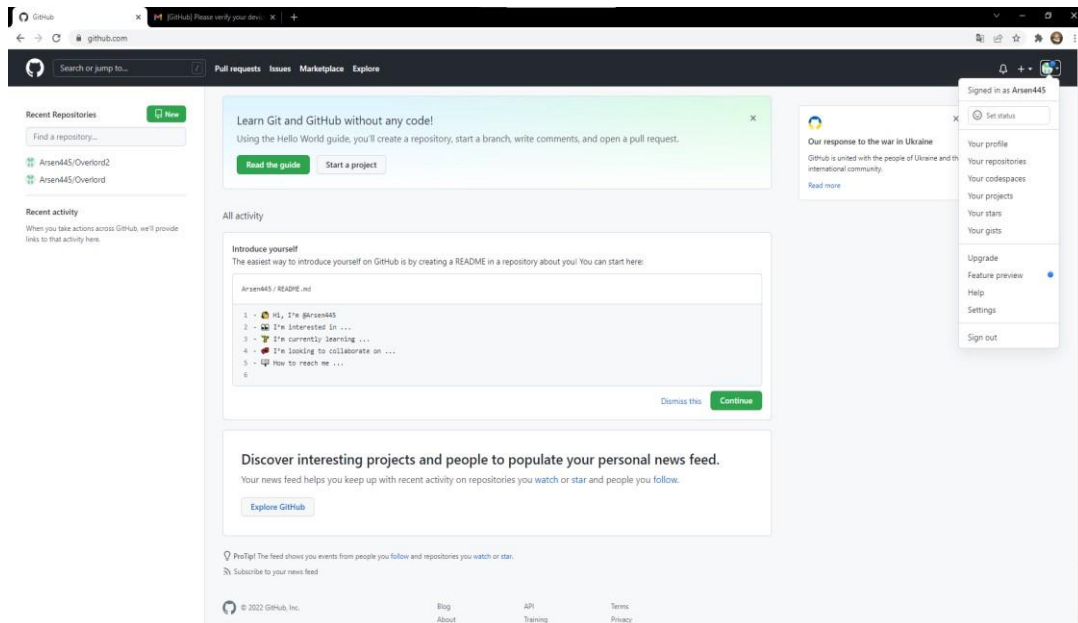
Лабораторная работа №1

Выполнил Эсеналиев Арсен

ИВТ-б-о-21-1

Цель: научиться пользоваться гитхабом.

1) Авторизируемся на гитхаб.



2) Устанавливаем гитхаб на ПК и проверяем был ли он установлен успешно.

```
Git CMD
C:\Users\GG_Force>git version
git version 2.35.1.windows.2
C:\Users\GG_Force>
```

3) Добавляем в Гит имя и адрес эл.почты.

```
C:\Users\GG_Force>git config --global user.name Arsen445
C:\Users\GG_Force>git config --global user.email arsen4545454@gmail.com
C:\Users\GG_Force>
```

4) Создаем новый репозиторий

Owner *

Arsen445 ▾

/


Repository name *

LB1.1 ✓


Great repository names are short and memorable. Need inspiration? How about [potential-octo-pancake?](#)

Description (optional)

LB1.1

☒  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: Python ▾

☒ 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

5) Клонировем репозиторий

```
C:\Users\GG_Force>git clone https://github.com/Arsen445/LB1.1.git
Cloning into 'LB1.1'...
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.

C:\Users\GG_Force>
```

6) Дополняем файл гитигнор

```
<> Edit file  Preview changes  Spaces 2 No wrap
19 downloads/
20 eggs/
21 .eggs/
22 lib/
23 lib64/
24 parts/
25 sdist/
26 var/
27 wheels/
28 share/python-wheels/
29 *.egg-info/
30 .installed.cfg
31 *.egg
32 MANIFEST
33
34 # PyInstaller
35 # Usually these files are written by a python script from a template
36 # before PyInstaller builds the exe, so as to inject date/other infos into it.
37 *.manifest
38 *.spec
39
40 # Installer logs
41 pip-log.txt
42 pip-delete-this-directory.txt
43
44 # Unit test / coverage reports
45 htmlcov/
46 .tox/
47 .nox/
48 .coverage
49 .coverage.*
50 .cache
51 nosetests.xml
52 coverage.xml
53 *.cover
54 *.n*.cover
```

7) Добавляем ридми

Arsen445 / LB1.1 Public

Pin Unwatch 1 Fork 0 Star 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights

main

Go to file Add file Code

Arsen445 Create README.md now 3

.gitignore Update .gitignore 2 minutes ago

LICENSE Initial commit 41 minutes ago

README.md Create README.md now

README.md

LB1.1

LB1.1 ИБТ-б-о-21-1 Эсеналиев Арсен

About

LB1.1

Readme

MIT License

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

8) Первый комит

The screenshot shows a code editor window titled 'ПРОГРАММА.py - D:\Ov...' with a menu bar (File, Edit, Format, Run) and a single line of Python code: `print('x y z w')`. To the right is a 'Git CMD' terminal window. It displays the command `git add .` being entered multiple times, followed by `git commit -m "Добавлен комит"`. The output shows the commit hash `8acb394` and indicates that 1 file changed with 1 insertion.

```
File Edit Format Run
print('x y z w')

The most similar command is
add

D:\Overlord>git add .
D:\Overlord>git add .
D:\Overlord>git add .
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>
D:\Overlord>git add .
D:\Overlord>git commit -m "Добавлен комит"
[main (root-commit) 8acb394] Добавлен комит
1 file changed, 1 insertion(+)
create mode 160000 LB1.1
D:\Overlord>
```

9) 2 коммит

The screenshot shows a code editor window titled 'asd.py - D:\LB1.1\asd.py (3.9.6)' with a menu bar (File, Edit, Format, Run, Options, Window, Help). The code contains two lines: `print('x y z w')` and `for y in 0, 1:`.

```
File Edit Format Run Options Window Help
print('x y z w')
for y in 0, 1:
```

10) 3 коммит

The screenshot shows a code editor window titled 'asd.py - D:\LB1.1\asd.py' with a menu bar (File, Edit, Format, Run) and two lines of code: `print('x y z w')` and `for y in 0, 1:` followed by `for x in 0, 1:`. To the right is a 'Git CMD' terminal window. It shows the command `git commit -m "Добавлен комит 3"` being entered. The output indicates that the branch is ahead of 'origin/main' by 1 commit. It also shows a warning about changes not staged for commit, followed by `git add .` and another `git commit -m "Добавлен комит 3"`. The final output shows the commit hash `0566c64` and indicates that 1 file changed with 2 insertions.

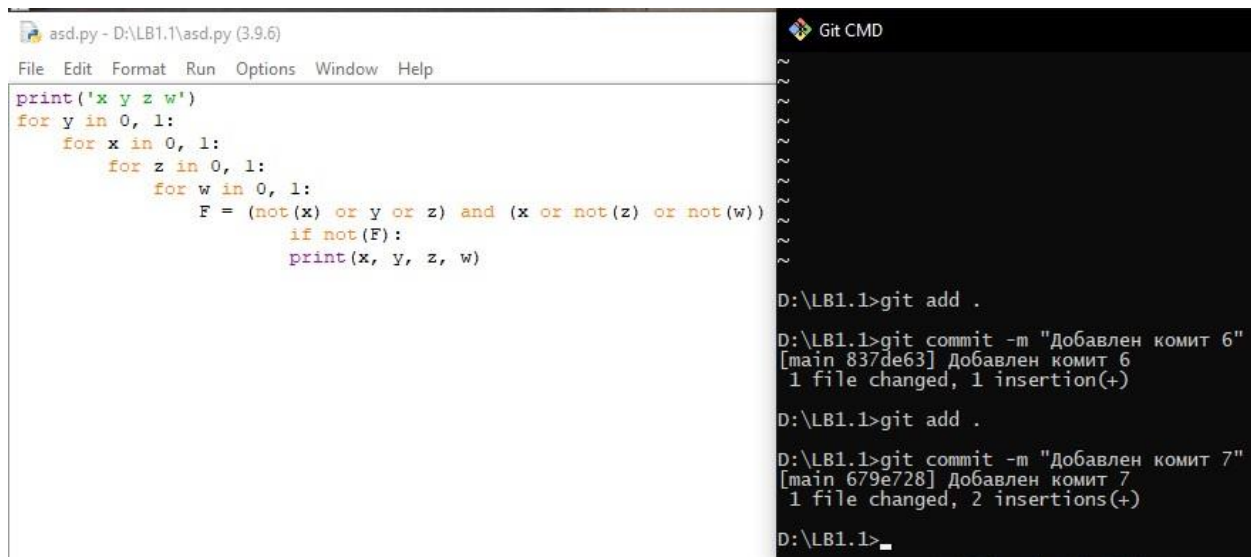
```
File Edit Format Run
print('x y z w')
for y in 0, 1:
for x in 0, 1:

D:\LB1.1>git commit -m "Добавлен комит 3"
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

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:   asd.py

no changes added to commit (use "git add" and/or "git commit -a")
D:\LB1.1>git add .
D:\LB1.1>git commit -m "Добавлен комит 3"
[main 0566c64] Добавлен комит 3
1 file changed, 2 insertions(+)
D:\LB1.1>
```

11) 4 коммит

The image shows two side-by-side windows. The left window is a Python IDE titled 'asd.py - D:\LB1.1\asd.py (3.9.6)'. It contains a Python script with nested loops for variables x, y, z, and w, and a logical expression F. The right window is a 'Git CMD' terminal. It shows the execution of 'git add .' followed by two 'git commit' commands with Russian messages, and another 'git add .' command.

```
asd.py - D:\LB1.1\asd.py (3.9.6)
File Edit Format Run Options Window Help
print('x y z w')
for y in 0, 1:
    for x in 0, 1:
        for z in 0, 1:
            for w in 0, 1:
                F = (not(x) or y or z) and (x or not(z) or not(w))
                if not(F):
                    print(x, y, z, w)

Git CMD
D:\LB1.1>git add .
D:\LB1.1>git commit -m "Добавлен комит 6"
[main 837de63] Добавлен комит 6
1 file changed, 1 insertion(+)
D:\LB1.1>git add .
D:\LB1.1>git commit -m "Добавлен комит 7"
[main 679e728] Добавлен комит 7
1 file changed, 2 insertions(+)
D:\LB1.1>
```

Контрольные 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 распределенным системам, поэтому не зависит от центрального сервера, где хранятся файлы.

4. В чем концептуальное отличие Git от других СКВ?

Git не хранит и не обрабатывает данные таким же способом как другие СКВ. Каждый раз, когда вы делаете коммит, т. е. сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Следует, что Git эффективен в хранении бэкапов, поэтому известно мало случаев, когда кто-то терял данные при его использовании.

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?

Зафиксированный файл – файл уже сохранён в вашей локальной базе.

Измененный файл – файл, который поменялся, но ещё не был зафиксирован.

Подготовленный файл — это изменённый файл, отмеченный для включения в следующий коммит.

7. Что такое профиль пользователя в GitHub?

Профиль – ваша публичная страница на GitHub, как и в социальных сетях. Когда мы ищем работу в качестве программиста, работодатели могут посмотреть наш профиль GitHub и принять его во внимание, когда будут решать, брать нас на работу или нет.

8. Какие бывают репозитории в GitHub?

Репозиторий Git бывает локальный и удалённый.

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 <YOUR_NAME>
```

```
git config --global user.email <EMAIL>
```

11. Опишите этапы создания репозитория в GitHub.

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

2) 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 репозиториев, 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.

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?

1) 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.

2) Откройте командную строку или терминал и перейдите в каталог, куда вы хотите скопировать хранилище. Затем напишите `git clone` и введите адрес.

14. Как проверить состояние локального репозитория Git?

Используя команду: `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` ?

Файлы обновятся на удалённом репозитории.

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) SmartGit 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: исследовал базовые возможности системы контроля версий Git и веб-сервиса для хостинга IT-проектов GitHub. Создал GitHub репозиторий, клонировал репозиторий на компьютер и отправил изменения на удалённый репозиторий.