Лабораторная работа №2.

Работа с Git и GitHub.

Цель работы:

Установка и работа в системе контроля версий Gir Bash, установка и регистрация на GitHub. Создание репозитория, клонирование. Команды GitHub'a.

1. Установка системы контроля версий Git.

В настоящее время владение Git стало обязательным требованием при приёме на работу не только для профессионалов, но даже для стажеров.

Git – это система контроля версий (СКВ). Существует несколько подобных систем, однако Git – на настоящий момент наиболее используемая СКВ.

Git создан для решения нескольких проблем любого программиста.

1) История изменений. Работа программиста – это всегда история изменений в коде программы.

a. Внося изменения в файлы, хочется знать ответ на вопросы «Кто сделал? Что сделал? Когда сделал?». Таким образом легко отслеживать, когда появились ошибки и кто их сделал. **Любая система, которая позволитнам видеть такую историю изменений, и является системой контроля версий.**

b. Иметь возможность отмены изменений или отката по истории назад (и вперёд).

2) Лёгкость внесения изменений.

a. Если нужно попробовать какой-то вариант – это должно быть просто.

b. Вернуться на основной вариант – также просто

c. Возможность легко принять или отвергнуть альтернативу.

3) Совместная работа

a. Хорошо тому живется, у кого одна нога… Если ног несколько, возникает проблема синхронизации.

b. Изменения разных пользователей нужно изолировать друг от друга…

c. … а по мере готовности – сливать вместе.

Программа Git решает все эти проблемы. Эта программа – набор скриптов, который умеет управлять изменениями. Он следит за файлами, ведет их историю, умеет ими манипулировать, откатывать, сливать и т.д.

Git используется в разработке ядра Linux и создан Линусом Торвальдсом.

Задача Git – вести полную историю изменений в некоей папке на сервере или локальном компьютере (**репозиторий**). К изменениям относятся добавление и удаление файлов, модификация их содержимого (нужно также учитывать, что Git не следит за пустыми папками, в папке нужно создать хотя бы один файл).

В историю также записывается, кто и когда сделал изменения.

Git – это распределенная система контроля версий, в ней нет центрального репозитория. Репозиторий – это просто папка с вашими файлами, в которой есть ещё некая служебная информация для Gita. Ихможет быть очень много. Репозитории могут обмениваться изменениями между собой. Однако, это не DropBox, он не занимается хранением файлов, его задача следующая. Он может от одного репозитория к другому передать изменения!

Установка Git.

Для Windows установки нужно перейти по ссылке.

https://git-scm.com/

Выбрать версию для Windows.

Нажимаем Download for Windows. Устанавливаем всё с настройками по умолчанию.

2. Создание локального Git-репозитория.

Система спроектирована как набор программ, специально разработанных с учётом их использования в сценариях. Это позволяет удобно создавать специализированные системы контроля версий на базе Git или пользовательские интерфейсы.

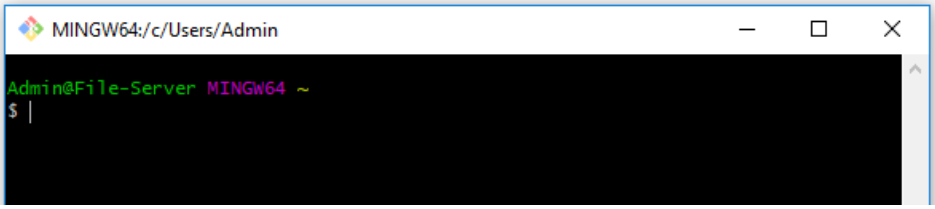
Git поддерживает быстрое разделение и слияние версий, включает инструменты для визуализации и навигации по нелинейной истории разработки. Как и Darcs, BitKeeper, Mercurial, Bazaar и Monotone[en], Git предоставляет каждому разработчику локальную копию всей истории разработки, изменения копируются из одного репозитория в другой.

Удалённый доступ к репозиториям Git обеспечивается git-демоном, SSH- или HTTP-сервером. TCP-сервис git-daemon входит в дистрибутив Git и является наряду с SSH наиболее распространённым и надёжным методом доступа. Метод доступа по HTTP, несмотря на ряд ограничений, очень популярен в контролируемых сетях, потому что позволяет использовать существующие конфигурации сетевых фильтров.

Создание репозитория.

1.Запустить GitBash из меню Пуск - Git.

Откроется следующая консоль:

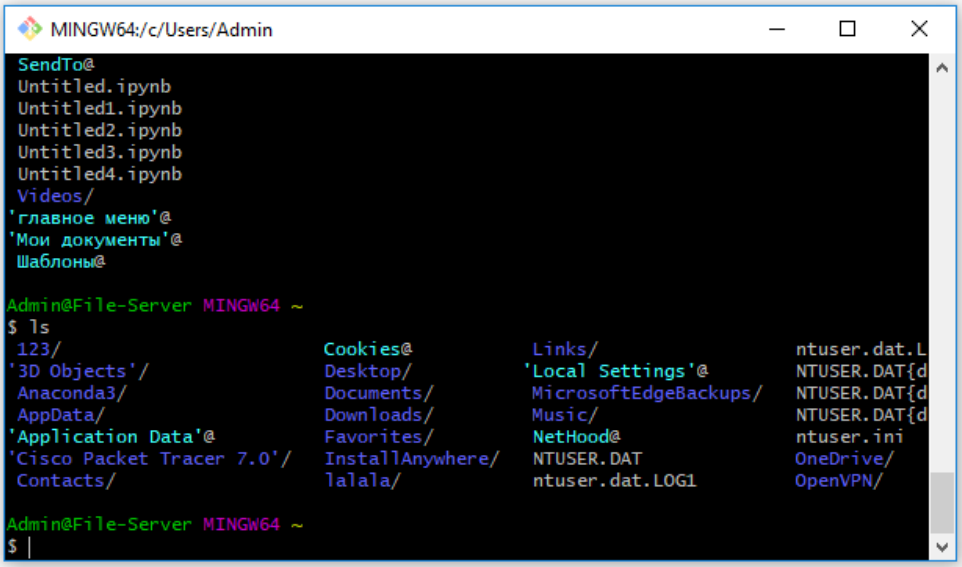


Это командная строка Linux (наподобие консоли командной строки Windows), аккуратно перенесенная в Windows.

Далее работа с Git будет объясняться на примере работы с консольным клиентом по следующим причинам:

- Чтобы у вас складывалось понимание происходящего и при возникновении проблем вы могли четко объяснить, что вы делали, и было видно, что пошло не так.
- Все нажатия кнопок в графических клиентах в итоге сводят к выполнению определенных команд консольного клиента, в то же время возможности графических клиентов ограничены по сравнению с консольным

2. Наберите команду ls. Если после этого вы получите список файлов и папок, значит, Bash успешно установился и запустился.

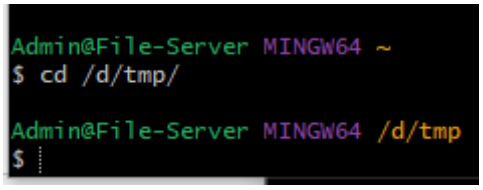


3. Далее для работы нам потребуется создать папку. Создайте её, например, в корне диска d (непосредственно из под Windows), назовите tmp. Как это сделать: вызываем командную строку в Windows командой cmd и создаем на диске пустую папку с помощью команды md “d: \tmp”.

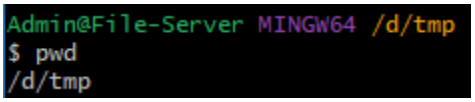
4. Теперь нужно в Bash перейти в эту папку. Для этого используем команду cd (change directory):

$ cd /d/tmp/

Нажмите Enter и вы окажетесь в этой папке. Если никаких сообщений об ошибке не выводится, значит, команда выполнена правильно.



5. Команда pwd показывает, какая директория текущая в данный момент. Наберите команду и проверьте, где вы находитесь.



6. Далее следует задать настройки Git. Они используются для того, чтобы отслеживать авторов изменений. На своем компьютере следует задать реальные имя, фамилию и email. Задайте имя «Пётр Иванов», «Ivanov@example.com», (используйте свои имя и фамилию).

Обратите внимание на то, что нажимая кнопку ↑ на клавиатуре, можно повторять ранее использованные команды, они будут выводиться в командной строке, после чего их можно редактировать и выполнять. Это значительно ускорит работу.

Попробуйте после ввода имени повторить команду и отредактировать её, задав адрес электронной почты.

```
Admin@File-Server MINGW64 /d/tmp
$ git config --global user.name "Пётр Иванов22"

Admin@File-Server MINGW64 /d/tmp
$ git config --global user.email "Ivanov22@example.com"

Admin@File-Server MINGW64 /d/tmp
$
```

Ключ *–global* означает, что для всех репозиториев будут действовать одни и те же настройки (если задать ключ --local или вообще не задать ключ, настройки будут храниться в данном репозитории и распространяться только на него).

7. Убедитесь, что вы находитесь в папке будущего репозитория (команда *pwd*). Выведите содержимое репозитория (команда *ls*). Убедитесь, что в данный момент папка пуста.

8. Дайте команду *git init*

Эта команда инициализирует репозиторий в текущей пустой папке, о чем выведется сообщение:

```
Admin@File-Server MINGW64 /d/tmp
$ git init
Initialized empty Git repository in D:/tmp/.git/
```

9. Выполните команду ls. Папка по прежнему пуста.

Теперь введите ту же команду с ключом ls -a:

```
Admin@File-Server MINGW64 /d/tmp (master)
$ ls

Admin@File-Server MINGW64 /d/tmp (master)
$ ls -a
./  ../  .git/
```

Вы видите, что в папке появились скрытые папки для служебных целей, созданные Git.

10. Попробуйте на диске D: создать пустую папку GitRepo и перенести туда данную папку tmp (сделайте это непосредственно из папки Мой компьютер, Bash можно не использовать). Далее в GitBash перейдите в эту папку (команда cd).

11. Проверьте, что скрытые файлы по-прежнему на месте, т.е. это по-прежнему репозиторий:

```
Admin@File-Server MINGW64 /d/tmp (master)
$ cd /d/GitRepo/tmp/

Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ pwd
/d/GitRepo/tmp

Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ ls

Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ ls -a
./  ../  .git/
```

12. Дайте команду *git status*.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Эта команда показывает, в каком состоянии в данный момент находится наш репозиторий.

В данном случае Git сообщает, что фиксировать нечего, изменений внутри репозитория не было. Т.о. к абсолютному пути репозиторий не привязан.

3. Просмотр истории коммитов.

Нижний уровень git является так называемой контентно-адресуемой файловой системой. Инструмент командной строки git содержит ряд команд по непосредственной манипуляции этим репозиторием на низком уровне.

Эти команды не нужны при нормальной работе с git как с системой контроля версий, но нужны для реализации сложных операций (ремонт повреждённого репозитория и так далее), а также дают возможность создать на базе репозитория git своё приложение.

Практическая часть.

История изменений.

1. Создайте в папке tmp файл README.txt.
2. Вызовите команду git status.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        README.txt

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

Мы видим, что появился неотслеживаемый файл (Untracked files). Т.е. в нашем репозитории появились посторонние файлы, которые Git видит, но пока не отслеживает.

3. Первое, что нужно сделать при таких изменениях – передать файл под контроль Git. Для этого используется команда git add.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git add README.txt
```

Будьте внимательны с регистром!

Если команда ничего не вывела, значит, она завершилась успешно.

4. Проверьте git status снова.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt
```

На этот раз Git видит этот файл, и сообщает, что он появился в репозитории. Эти изменения ещё не зафиксированы, но они уже проиндексированы. Т.е. Этот файл готов к тому, чтобы быть зафиксированным.

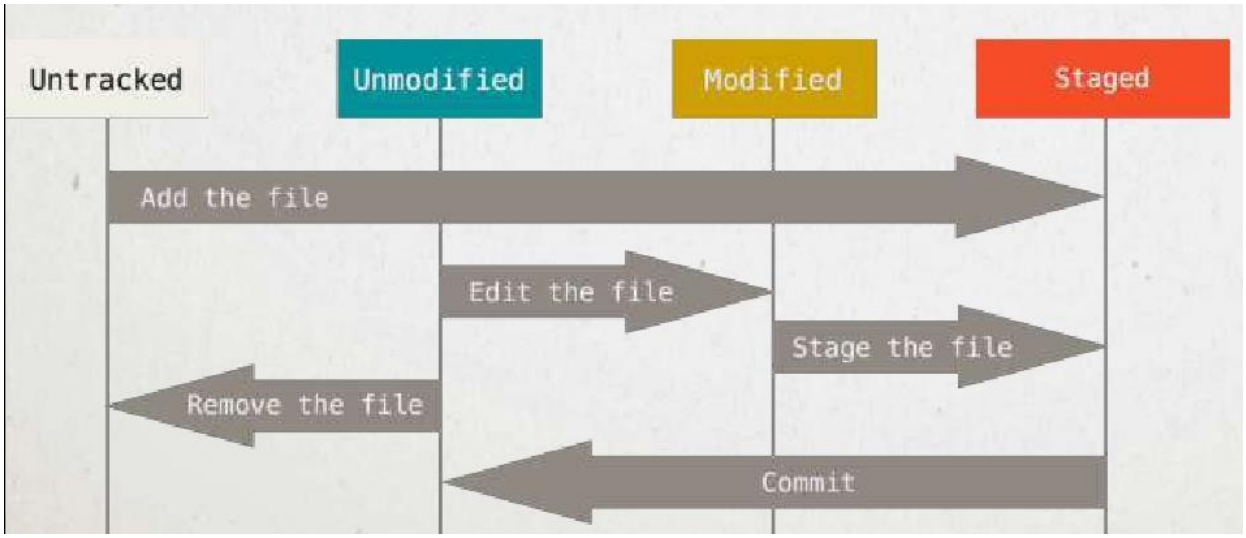
5. Следующий шаг, который мы должны сделать, передав файл под контроль Git и закончив все изменения в нём – закоммитить их (фиксировать изменения).

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git commit -m "Добавлен файл README"
[master (root-commit) 2519868] Добавлен файл README
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.txt
```

Команду git commit следует ОБЯЗАТЕЛЬНО использовать с ключом –m и комментарием!

Рассмотрим полученное от Git сообщение. Тут написано, что один файл изменён.

Итак, в Git файлы могут находиться в четырёх состояниях:



1) Untracked – файл просто находится в папке, но для Git он неизвестен, его в истории нет и никогда не было.

2) Staged – подготовленный для фиксации изменений (командой add). Это значит, что файл под контролем Git и он ждёт нашей команды, чтобы зафиксировать изменения (т.е. установить флажок в истории с комментарием).

3) Unmodified–означает, что файл со времени последней фиксации не менялся. Переводится в это состояние фиксацией (commit).

4) Modified – измененный файл. Этот файл уже был в истории, Git о нём знает, файл участвовал в каких-то коммитах, но мы его сейчас изменили. Теперь мы можем либо перевести его в Staged (add), либо отменить изменения.

Наша задача – отчётливо понимать, как происходит перемещение файла по этим состояниям. Результатом работы должно быть состояние файлов Unmodified.

5. Проверьте git status снова. В данный момент все должно быть Unmodified – nothing to commit.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master
nothing to commit, working tree clean
```

6. Откройте файл README.txt и напишите в нем «Hello, world!». Сохраните файл. Проверьте git status снова.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Git сообщает о том, что файл был изменён. И он не готов к фиксации (changes not staged for commit).

Каким образом Git это определяет? У него в папке хранятся идеальные копии наших файлов, и он очень быстро их сравнивает с реальными файлами.

7. Готовим изменения к фиксации:

Git add README.txt

8. Проверяем текущий статус.

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git add README.txt

Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.txt
```

Видим, что измененный файл *README.txt* готов к фиксации.

9. Далее следует зафиксировать изменения (commit). Не забудьте про ключ и комментарий!!!

```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git commit -m "Изменения в README"
[master 49f0cfd] Изменения в README
 1 file changed, 1 insertion(+)
```

Дополнительные команды:

Git rm FILE–удаляет файл из индекса и из рабочей папки; *git rm –cached FILE* – удаляет файл только из индекса Git.Оба этих удаления требуют фиксации!

10. Для того, чтобы просмотреть историю коммитов, нужно использовать команду git log – это лог всех изменений.

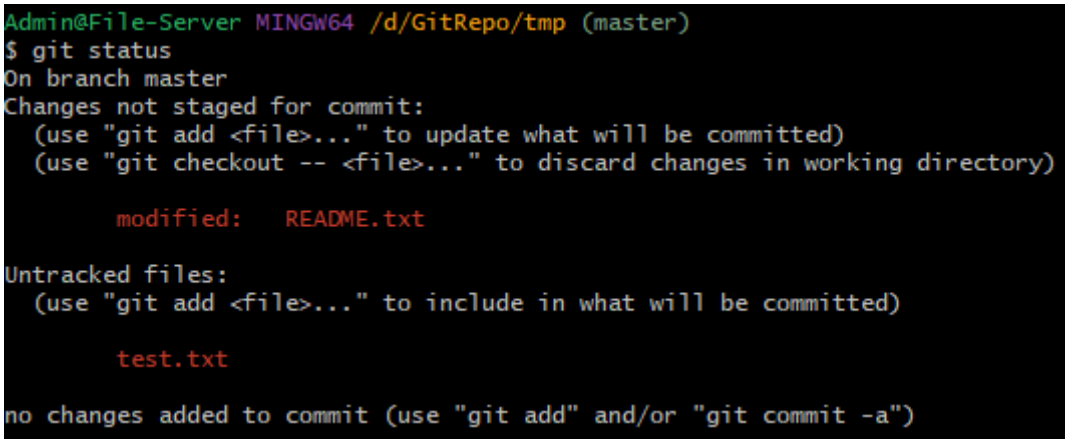
```
Admin@File-Server MINGW64 /d/GitRepo/tmp (master)
$ git log
commit 49f0cfdb330fd2d8a4319f16a39e3cb55f0e4b11 (HEAD -> master)
Author: Пётр Иванов22 <Ivanov22@example.com>
Date:   Tue Aug 14 09:34:33 2018 +0300

    Изменения в README

commit 251986888c8f25d0f2c28bc0102b64c0f3cff567
Author: Пётр Иванов22 <Ivanov22@example.com>
Date:   Tue Aug 14 09:00:59 2018 +0300

    Добавлен файл README
```

Здесь мы видим созданные нами два коммита. Длинные числа из шестнадцатеричных цифр – это их номера (кэш). Такие длинные номера нужны для того, чтобы они были уникальными. Они непоследовательны, это сделано специально, для веток.

К каждому коммиту указан автор и дата/время изменений.

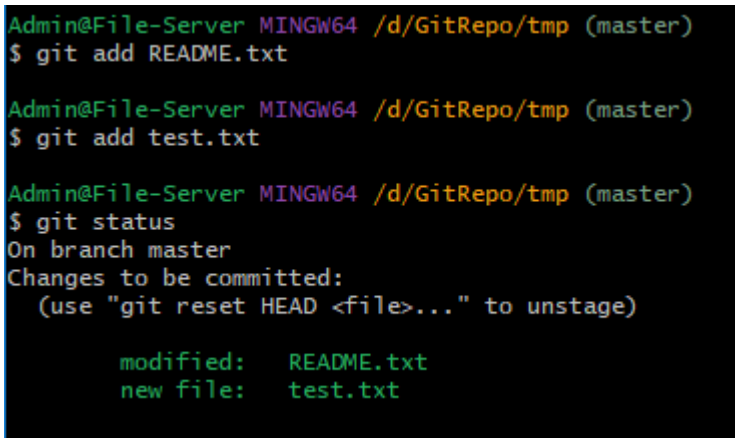
11. Создайте в нашей папке ещё один файл test.txt, напишите внутри него TEST. В файле README добавьте пару восклицательных знаков или что-либо ещё. Таким образом мы получили два изменения в репозитории.

12. Проверьте текущий статус.

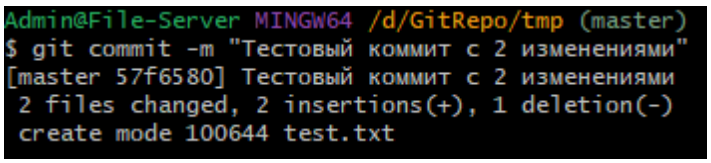


Мы видим сообщение о том, что два изменения дожидаются **стейджинга**, т.е. индексации. Для чего вообще выполняется индексация? Таким образом мы показываем, что изменения в данном файле важны и мы планируем их зафиксировать. Если файл не проиндексирован, значит, мы продолжаем его обрабатывать и не уверены, что будем сохранять изменения.

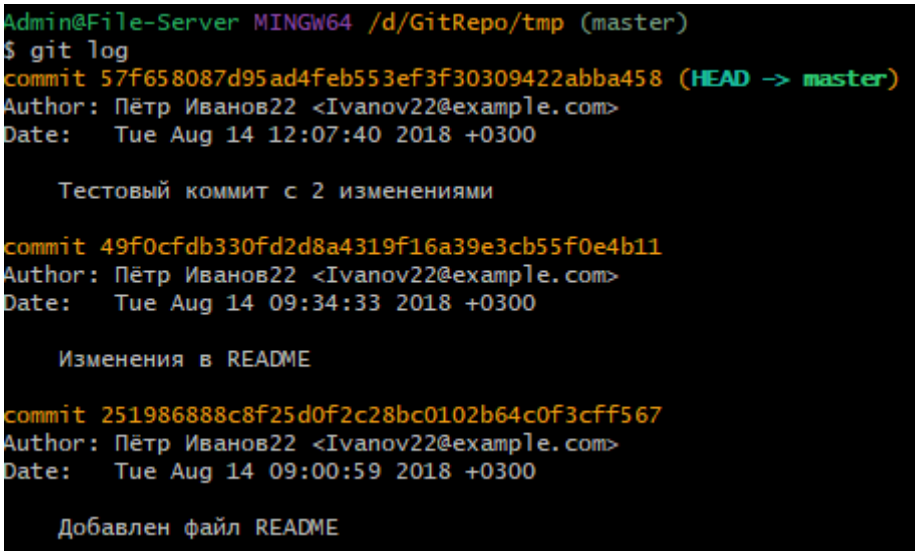
13. Добавляем (индексируем) оба файла:



14. Зафиксируем оба изменения в одном коммите:



Просмотрите историю коммитов в git log. Там появился новый только что созданный коммит.



Чтобы более подробно видеть информацию о коммитах, в gitlog можно использовать ключ -p. В этом случае по каждому изменению будет выведено, что было и что стало в каждом измененном файле.

4. Работа с GitHub.

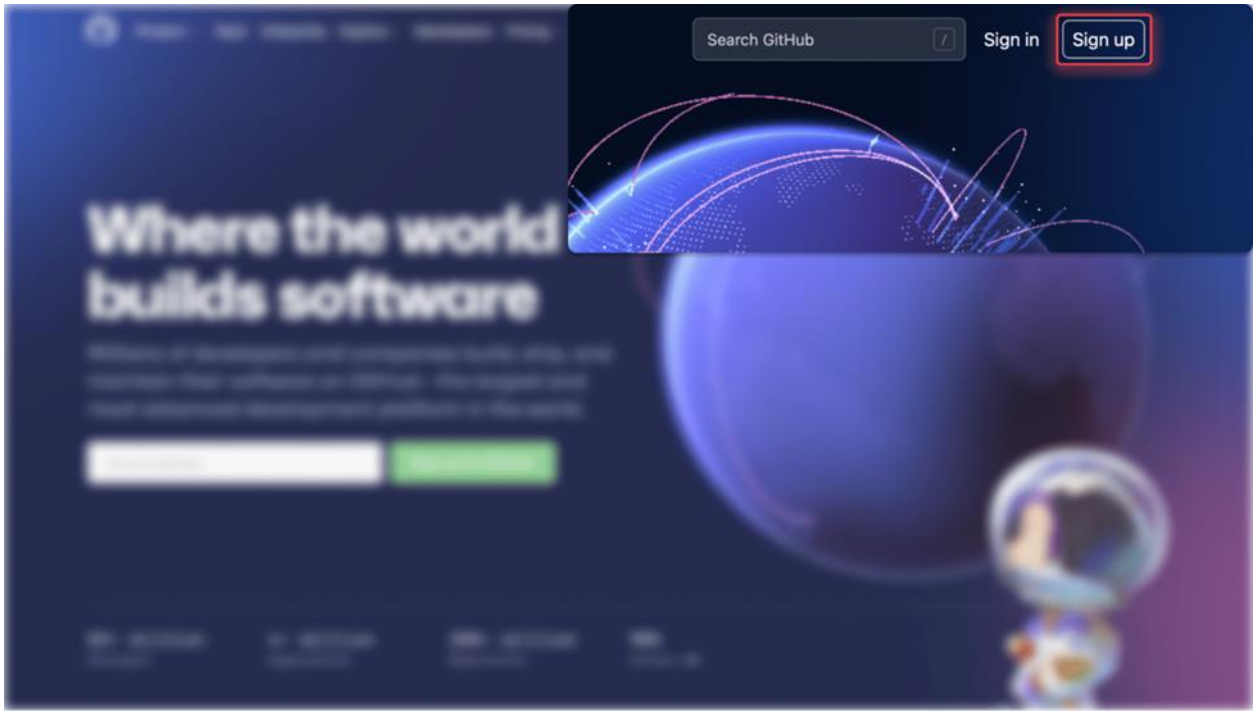
GitHub — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан компанией GitHub, Inc. Википедия

На его примере разберем, как работать с получением изменений и их отправкой.

Регистрация на GitHub

GitHub — веб-сервис, который основан на системе Git. Это такая социальная сеть для разработчиков, которая помогает удобно вести коллективную разработку IT-проектов. Здесь можно публиковать и редактировать свой код, комментировать чужие наработки, следить за новостями других пользователей.

Для работы с платформой нужно создать аккаунт. Для этого переходим по ссылке: https://GitHub.com и тапаем по кнопке Sign up.

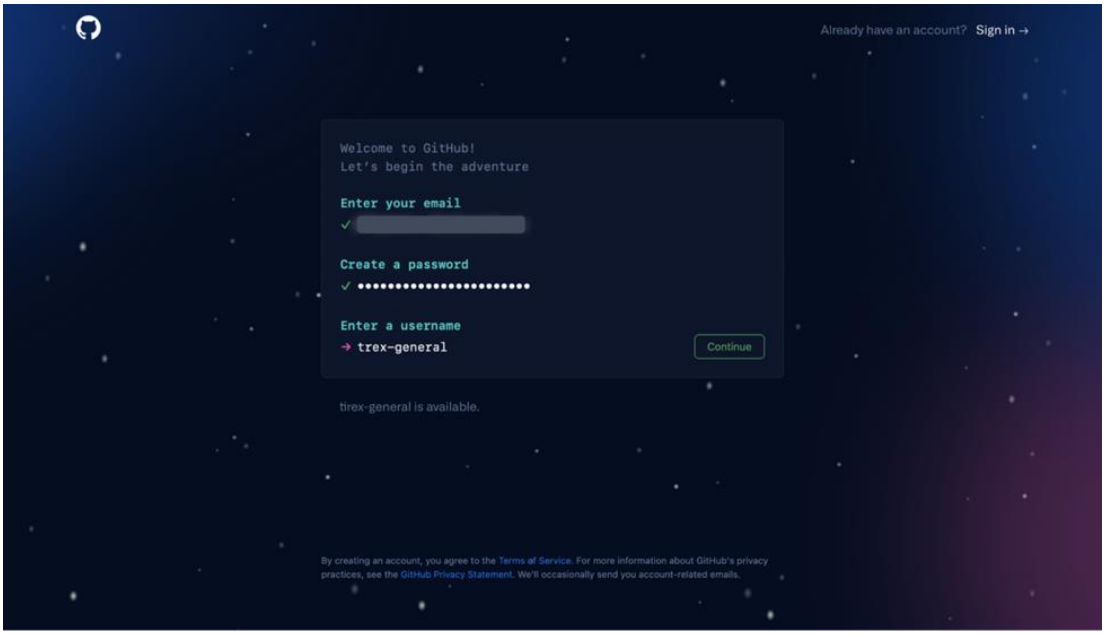


На странице регистрации вводим данные:

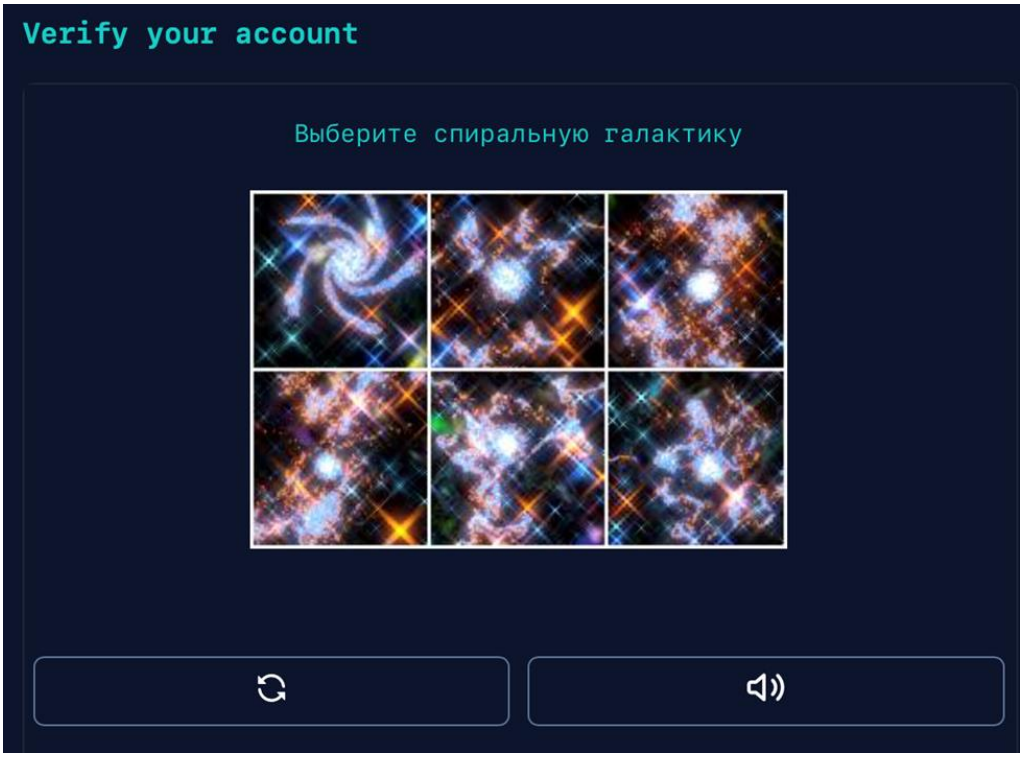
Адрес электронной почты. Если на почту уже был зарегистрирован аккаунт, на странице появится сообщение об ошибке: «Email is invalid or already taken».

Пароль. Система рекомендует использовать для пароля последовательность из 15 символов или 8, но с использованием хотя бы одной цифры и строчной буквы.

Имя пользователя. «Юзернейм» должен быть уникальным. При этом он не может начинаться или заканчиваться дефисом.



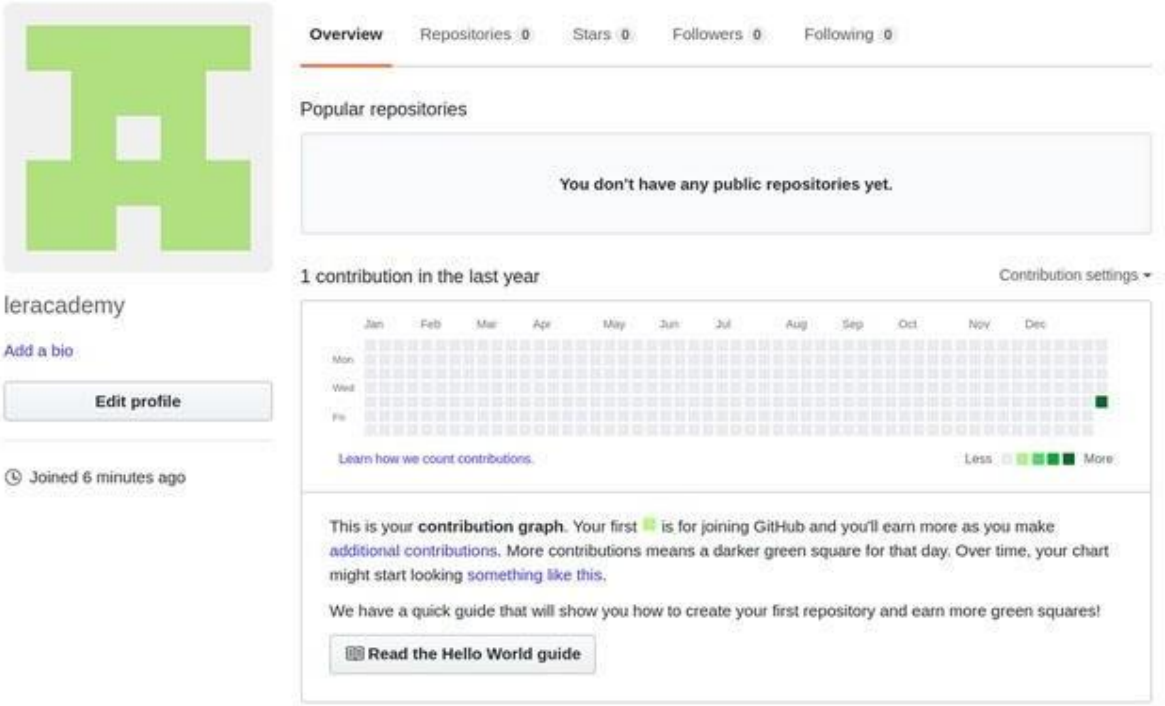
Теперь нужно нажать кнопку Continue, принять или отклонить предложение о подписке на рассылку и пройти экстравагантную валидацию:



Затем подтвердите адрес электронной почты.

Во всплывающих окнах указывайте настройки на свое усмотрение. Для ознакомления и некоммерческого использования достаточно бесплатного тарифа. Регистрация завершена.

Переход в ваш профиль.



Так выглядит ваш профиль после регистрации.

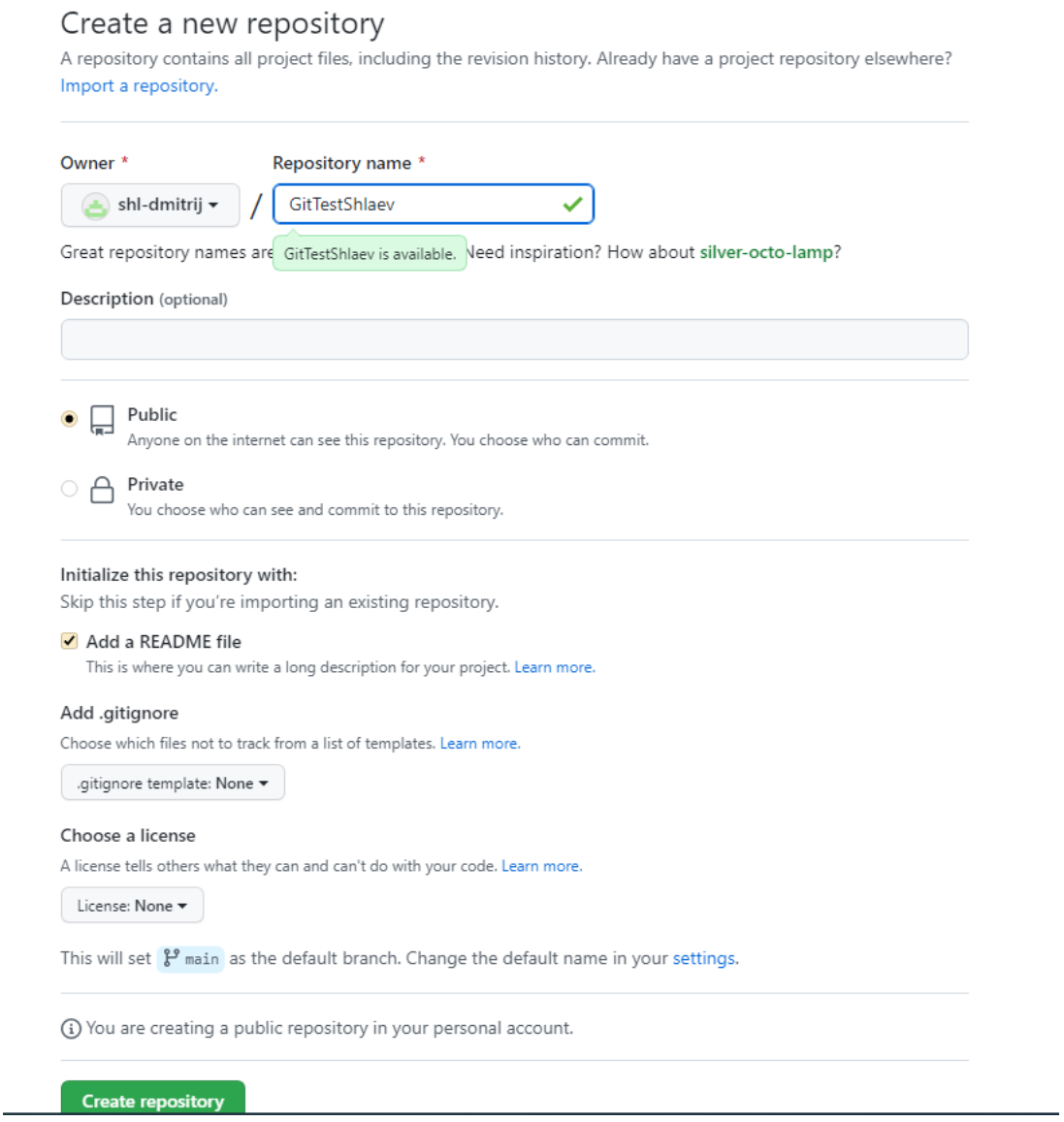
Теперь у вас есть профиль на GitHub.

Сейчас у нас нет ни одного репозитория, и мы можем либо создать новый репозиторий, либо ответвиться (fork) от уже существующего чужого репозитория и вести собственную ветку разработки. Затем, при желании, свои изменения можно предложить автору исходного репозитория (Pullrequest).

7.Если Вы находитесь в своем профиле, то в верхней части окна видите один из пунктов меню «Repositories 0». Это означает, что у вас в данный момент нет созданных репозиториев. На этой же вкладке есть кнопка

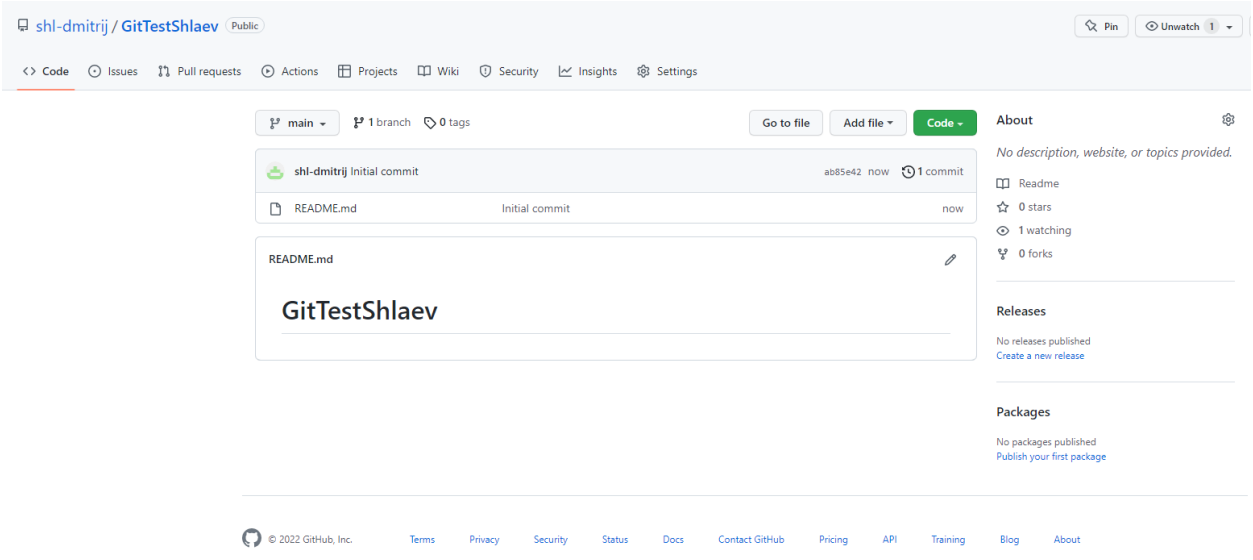
«New», которая позволяет создать новый репозиторий. Нажимаем её.

8.Открылось окно создания репозитория. Даем ему имя (например, GitTest) можно написать описание. Он должен быть публичным. Также установите галочку там, где предлагается поместить в репозиторий текстовый файл Readme. Нажмите кнопку «Create repository».

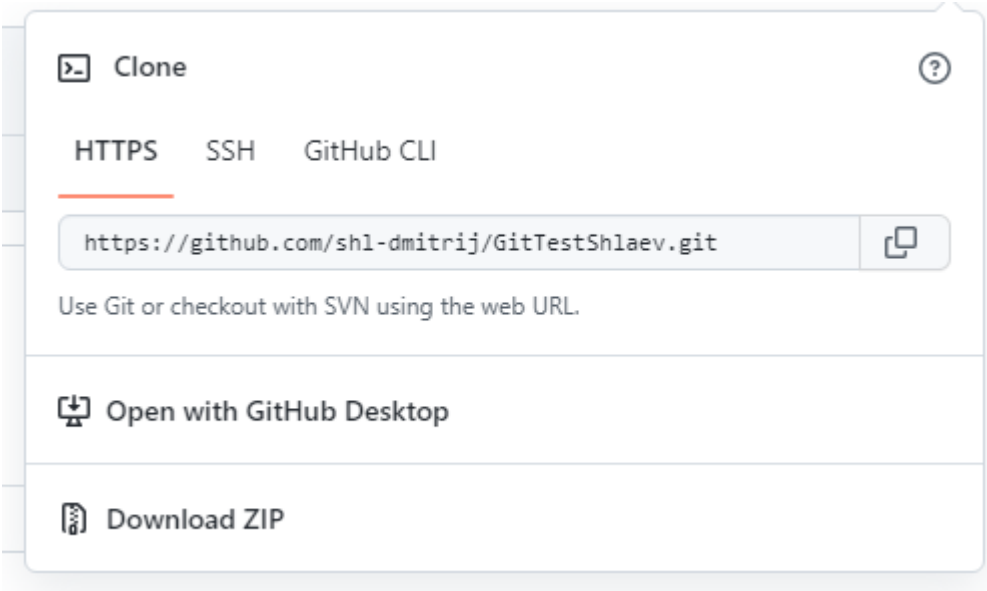


Когда вы создаете новый репозиторий таким образом, на GitHub появляется папка и для нее выполняется команда gitinit, т.е. происходит то же самое, что мы делали выше в вручную.

10.После создания репозитория вы в него попадаете и видите, что в нем в данный момент находится один файл – README.md, который был создан автоматически:



11. Далее нам нужно скопировать в буфер обмена адрес нашего репозитория на GitHub. Для этого нажмите кнопку «Cod»и в открывшемся окне копку справа от адреса ссылки:



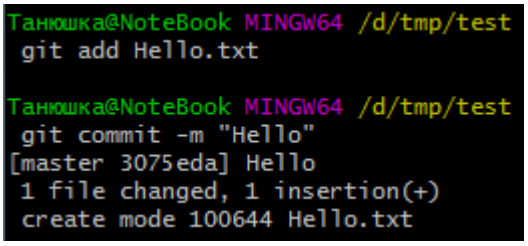
12. Откройте консоль GitBash (она должна быть у вас открыта, если нет – перейдите в папку tmp, с которой мы работали). Даем команду

git clone https://GitHub.com/DrovosekovaT/GitTestDrovosekova.git test (у каждого своя)

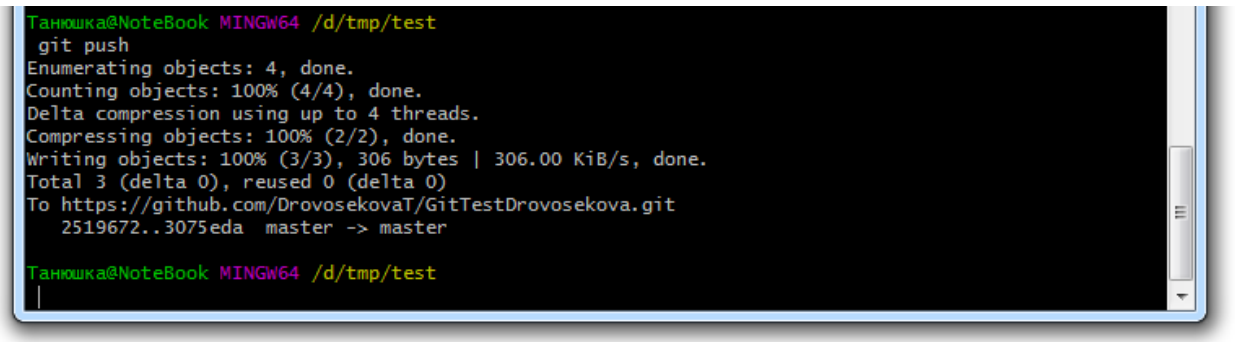
(вставьте из буфера обмена свой адрес репозитория и после него через пробел название папки, где будет создан клон удаленной папки-репозитория, в данном случае test).

Эта команда создаст в папке test копию репозитория с указанного адреса с GitHub, автоматически GitHub подключит к локальному репозиторию как удалённый и назовёт его origin, т.е. оригинал (перейдите в папку test и проверьте это командой git remote -v), автоматически в нашем локальном репозитории создаст локальную ветку master (проверьте это командой git gstatus), которую свяжет с веткой master в удаленном репозитории на GitHub.

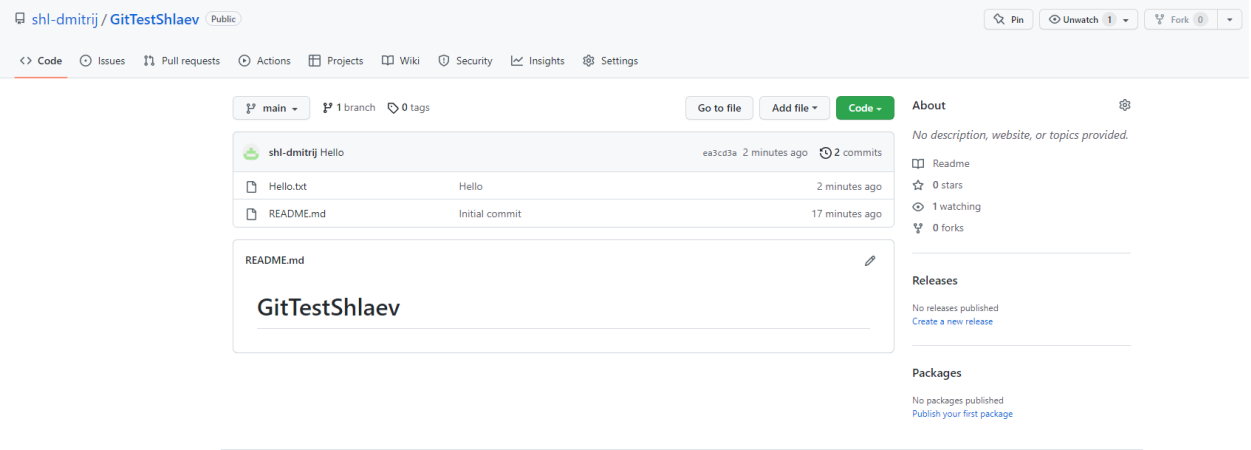
13. Теперь рассмотрим, как синхронизировать локальный репозиторий (клон) с оригиналом, который находится на GitHub. В клонированной папке test создайте локально файл Hello.txt, напишите в нем текст «Hello!!!», закройте его, сохранив изменения. Теперь у нас репозиторий содержит 2 файла, а на GitHub только один. Для того, чтобы зафиксировать изменения, выполните команду git add Hello.txt и создайте коммит:



14.Выполните команду git push. Эта команда берет имеющиеся локальные изменения и отправляет их в удалённый репозиторий. В ответ на запрос введите своё имя на GitHub и пароль.



15.Откройте репозиторий на сайте GitHub, убедитесь, что файл добавился:



Если вы откроете добавившийся файл Hello.txt, то увидите свой текст.

Также вы видите название коммита во втором столбце.

Итоги:

gitpull – не просто «достает изменения» из удаленного репозитория, но и пытается применить изменения из удалённого репозитория к нашему (влить удалённую ветку в локальную). Ветки будут изучаться далее.

gitpush – отправляет наши наборы изменений на удалёный репозиторий.

Таким образом, для чего применяется GitHub в реальной работе?

1. GitHub очень удобно использовать для быстрого бесплатного создания репозитория в интернете.

Для чего может понадобиться репозиторий в интернете?

1.Дать кому-то на него ссылку, чтобы этот человек мог удобно посмотреть на код, хранящийся там, удобно посмотрел на историю изменений.

2.GitHub может служить посредником для коллективной работы.

3. Если вы работаете за несколькими компьютерами, можно клонировать на них репозиторий с GitHub и синхронизировать через него все версии.

Задание для самостоятельной работы

1.Создайте у себя на компьютере еще один репозиторий ("второй").

2. Свяжите его с локальным. так же созданным вторым, получив результат, произведите его коммит.

3. Создайте файл в блокноте скопируйте из Git Bush результат и разместите файл на платформе для проверки преподавателем.