Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №1**

Дисциплина: «Разработка профессиональных приложений»

Обработка списков

Вариант №8

Выполнил:

студент группы ИВТАСбд-21

Зюзин Г.А.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Техническое задание.**

Познакомиться с технологией Git, с его базовыми функциями, понять принцип работы данной технологии. Научиться применять данные навыки работы на примере создания локального репозитория, добавления в него файлов, игнорирование файлов, создание нескольких веток, внесения изменений в файлы, клонирование репозитория в папку (в отличную от оригинальной в том числе), решать конфликты git.

**Ход работы.**

Git – это распределённая система управления версиями, которая используется для отслеживания изменений в исходном коде программы. Git позволяет разработчикам работать совместно над проектами, отслеживать изменения, вносить исправления и добавлять новые функции.

*1. Создание локального репозитория.*

Локальный репозиторий – это хранилище, расположенное на локальной машине, которая используется для хранения и управления версиями исходного кода проекта.

Чтобы создать локальный репозиторий, можно использовать несколько существующих способов. В этом отчёте будет описано два таких способа:

1. Через терминал Windows;
2. Через Git Bash.
3. Если уже есть проект в каталоге, который не находится под версионным контролем Git, то для начала нужно перейти в него и нужно ввести команду git init. Для Windows:

|  |
| --- |
| PS C:\Users\georg> cd C:\Users\georg\PycharmProjects\test  PS C:\Users\georg\PycharmProjects\test> git init |

1. Либо через каталог с проектом открываем Git Bash и вводим git init:

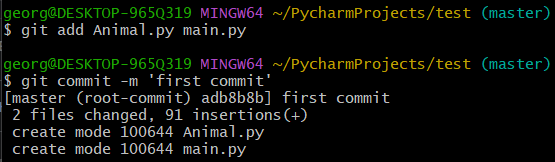
|  |
| --- |
| $ git init |

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git, содержащий все необходимые файлы репозитория — структуру Git репозитория.

*2. Добавить несколько файлов*

Чтобы добавить под версионный контроль существующие файлы (в отличие от пустого каталога), стоит добавить их в индекс и осуществить первый коммит изменений. Добиться можно запустив команду git add, указав индексируемые файлы (или загрузить сразу все файлы с помощью $ git add .), а затем выполнив git commit:

|  |
| --- |
| $ git add .  $ git commit -m 'first commit' |

****Если необходимо добавить несколько файлов, но не все, то $git add <filename1> <filename2> ...:

*3. Сделать один из файлов игнорируемым*

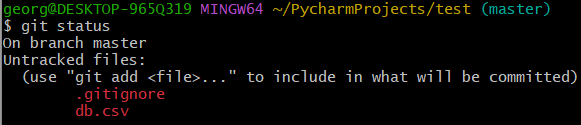
Для игнорирования файла в Git необходимо создать файл .gitignore в корневой директории репозитория и указать в нём названия файлов или директорий, которые нужно игнорировать. Данная команды может быть использована, например, если проект подразумевает работу с личными данными, которые не стоит фиксировать для сохранения конфиденциальности.

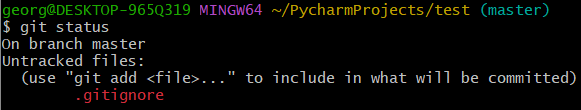
Необходимо создать файл .gitignore в корневой директории репозитория. Чтобы создать этот файл в Bash:

|  |
| --- |
| touch .gitignore |

В файл .gitignore в любом текстовом редакторе и добавляем название файла, который мы хотим игнорировать, например db.csv. И сохраняем изменения.

Теперь, когда выполняется команда git status, файл db.csv будет проигнорирован Git и не будет отображаться в списке проиндексированных файлов:

  
Рис. – до добавления db.csv в .gitignore.

  
Рис. – после добавления db.csv в .gitignore.

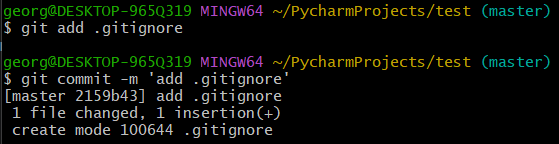
*4. Зафиксировать изменения*

После добавления файла db.csv в .gitignore сам .gitignore не является игнорируемым файлом, поэтому необходимо зафиксировать измения в Git.

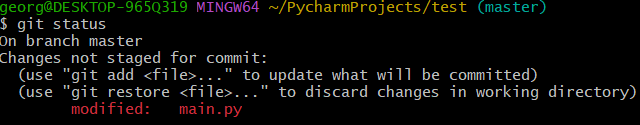
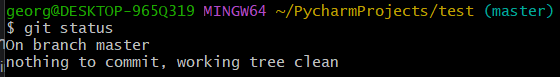
С помощью команды git add добавляем в индексацию новые файлы. После с помощью команды git commit фиксируем изменения:

|  |
| --- |
| $ git add .gitignore  $ git commit -m 'add .gitignore' |

В «коммитах» указываются описание внесённых изменений, для того чтобы можно было понять, что было изменено.

 После выполнения этих команд изменения будут зафиксированы в локальном репозитории.

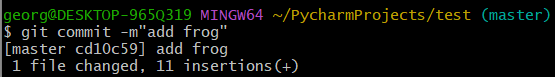
1. *Внести изменения в файлы и зафиксировать изменения.*

**До того как внести какие-либо изменения в файлах, команда git status выдавало следующее:  
 После изменения файла main.py команда git status выдаёт следующее:

Необходимо снова проиндексировать уже измененные файлы с помощью git add <filename> (git add .).

|  |
| --- |
| git add main.py |

Здесь main.py - это файл (он может быть не один), который были изменен и который нужно добавить в индексацию. Если изменения были внесены во все файлы, то можно вместо списка файлов указать . для добавления всех измененных файлов.

 И зафиксировать изменения с помощью git сommit -m:

1. *Создание нескольких бранчей, редактирование файлов в разных бранчах и фиксировать изменения. Слияние веток.*

В Git ветка (branch) представляет из себя указатель на коммит. Каждый раз, когда создаётся новая ветка, Git создает указатель, указывающий на текущий коммит, и помещает его в файл .git/refs/heads/. Когда производится фиксирование изменений, Git сохраняет состояние проекта и сохраняет указатель на этот коммит в ветке. Ветки позволяют одновременно работать над несколькими изменениями, не влияя на основную ветку.

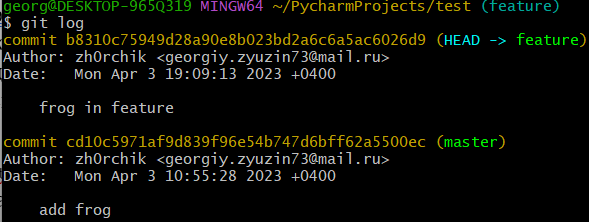
Чтобы создать новую ветку в Git, необходимо использовать команду git branch <branch\_name>, где <branch\_name> - название новой ветки. Например, для создания ветки с названием "feature" необходимо выполнить команду:

|  |
| --- |
| $ git branch feature |

После создания новой ветки можно переключиться на нее, используя команду git checkout <branch\_name>, где <branch\_name> - название ветки, на которую нужно переключиться. Например, для переключения на ветку "feature" необходимо выполнить команду:

|  |
| --- |
| $ git checkout feature  Switched to branch 'feature' |

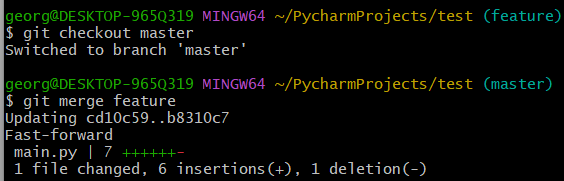
После переключения на новую ветку можно внести изменения в файлы и зафиксировать изменения, используя команды git add и git commit, как описано выше.

Чтобы проверить, что указатель находится на ветке "feature", можно ввести git log:

Последний коммит сделан в ветке "feature", а предыдущие в основной - "master".

Чтобы осуществить слияние двух веток, необходимо выполнить команду git merge, указав ветки, которые нужно слить. Например, для слияния ветки "feature" с веткой "master" нужно выполнить команду:

|  |
| --- |
| $ git checkout master  $ git merge feature |

После этого все изменения из ветки "feature" будут добавлены в ветку "master".

1. *Отправить исходные коды в удаленный репозиторий.*

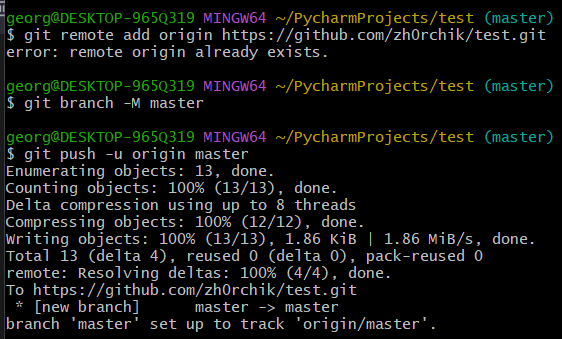
Удаленный репозиторий - это репозиторий Git, который находится на удаленном сервере, обычно на специально предназначенной для этого платформе, такой как GitHub, GitLab или Bitbucket. Удаленный репозиторий обычно служит для совместной работы над проектом, когда несколько разработчиков могут работать над одним и тем же кодом и синхронизировать свои изменения в общем репозитории.

Чтобы отправить исходные коды в удаленный репозиторий нужно выполнить следующие шаги:

1. Добавить удалённый репозиторий в список доступных для локального репозитория. Для этого используется команда git remote add <имя> <url>, где <имя> - это имя удаленного репозитория (например, "origin"), а <url> - URL-адрес удаленного репозитория.
2. Зафиксировать все изменения в локальном репозитории с помощью команды git commit.
3. Отправить на удалённый репозиторий с помощь команды git push <имя> <ветка>, где <имя> - имя удаленного репозитория, а <ветка> - название ветки, которую нужно отправить. Если ветка не указана, то по умолчанию будет отправлена текущая ветка.

Чтобы отправить исходные коды на удалённый репозиторий "origin" в ветку "master":

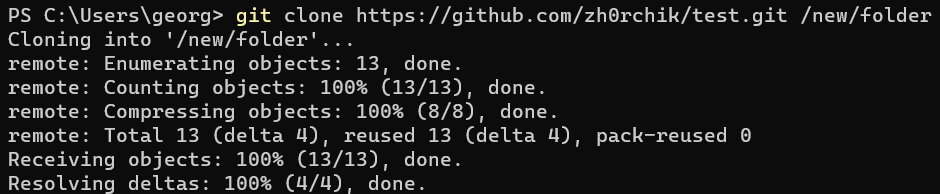
|  |
| --- |
| $ git remote add origin https://github.com/zh0rchik/test.git  $ git branch -M master  $ git push -u origin master |

* git remote add origin https://github.com/zh0rchik/test.git: Команда добавляет удаленный репозиторий с названием "origin" и адресом https://github.com/zh0rchik/test.git в локальный репозиторий Git.
* git branch -M master: Команда переименовывает текущую ветку в "master". Опция -M означает "force rename" и переименовывает текущую ветку даже если уже существует ветка с таким именем.
* git push -u origin master: Команда отправляет изменения из локального репозитория на удаленный репозиторий "origin" в ветку "master". Опция -u означает "set upstream" и устанавливает соответствие между локальной веткой "master" и удаленной веткой "master" на удаленном репозитории"origin". После этого можно использовать команду git push без указания имени ветки и удаленного репозитория, чтобы отправлять изменения на удаленный репозиторий.

После того, как вы выполнили эти команды, локальный репозиторий будет связан с удаленным репозиторием на GitHub. При выполнении команды git push -u origin master вы отправите все изменения из локального репозитория в удаленный репозиторий на ветку master. В дальнейшем, если вы будете делать изменения и коммиты в локальном репозитории, можно отправлять их в удаленный репозиторий с помощью команды git push. Если же вы захотите получить изменения, которые были сделаны в удаленном репозитории, вы можете выполнить команду git pull.

Все коммиты, которые были сделаны в локальном репозитории так же отправляются на удалённый репозиторий.

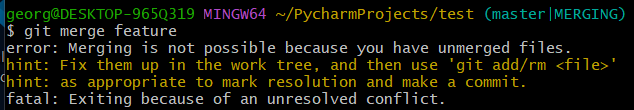
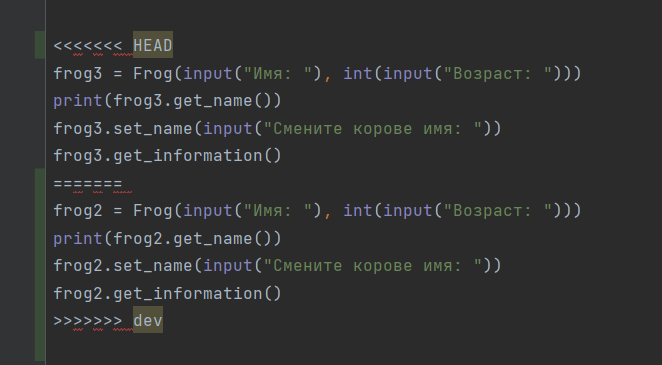
1. *Клонирование удаленный репозиторий в папку, отличную от оригинальной.*

**Для клонирования удаленного репозитория Git в папку, отличную от оригинальной, можно использовать команду git clone и указать путь до папки, в которую нужно склонировать репозиторий. Например, если нужно склонировать репозиторий по адресу https://github.com/zh0rchik/test.git в папку /new/folder, можно выполнить следующую команду в командной строке. В Windows клонировать репозиторий необходимо через терминал, так как может быть ошибка "Permission denied" указывает на то, что у вас нет прав на запись в указанную папку /new\_test.:

1. *Добиться конфликта файлов при применении изменений в оригинальном репозитории.*

Конфликт в Git - это ситуация, когда два или более пользователей одновременно изменяют один и тот же файл в разных ветках, и при попытке слияния этих веток Git не может автоматически решить, какое изменение следует принять. В этом случае Git создает конфликтный файл, который нужно разрешить вручную, чтобы восстановить целостность репозитория. Конфликт может возникнуть не только при слиянии веток, но и при слиянии изменений из удаленного репозитория, при перебазировании ветки и в других случаях.

Для создания конфликта файлов в оригинальном репозитории нужно выполнить следующие шаги:

1. Склонировать репозиторий на локальную машину с помощью команды git clone <url>.
2. Создать новую ветку и переключиться на нее с помощью команды git checkout -b <branch\_name>.
3. Внести изменения в файлы проекта в созданной ветке и закоммитить их с помощью команд git add <file> и git commit -m "<commit\_message>".
4. Переключиться на ветку master с помощью команды git checkout master.
5. Внести изменения в те же файлы проекта, которые были изменены в созданной ветке, и закоммитить их с помощью команд git add <file> и git commit -m "<commit\_message>".
6. Попытаться слить изменения из созданной ветки в ветку master с помощью команды git merge <branch\_name>.
7. *Решить конфликт.*
8. Git выдаст сообщение о конфликте файлов. Их нужно решить вручную, внести необходимые изменения в код с помощью текстового редактора.
9. После решения конфликтов нужно закоммитить изменения с помощью команд git add <file> и git commit -m "<commit\_message>".
10. После этого можно отправить изменения в удаленный репозиторий с помощью команды git push.

**Вывод.**

В работе были выполнены задачи: создание локального репозитория, добавление, игнорирование, фиксация и изменение файлов, создание и слияние веток, отправка и клонирование удаленного репозитория, создание и разрешение конфликта файлов. Получены навыки работы с Git: управление репозиторием, переключение веток, слияние, работа с удаленным репозиторием и разрешение конфликтов. Были изучены основы Git и приобретены практические навыки.