Лабораторная работа 7. Ветки

**Цель**. Научиться работать с ветками в git.

**Замечание**. Не забывайте перед каждым коммитом сохранять изменения!!!

**Задание**. Погодный сервис (openweathermap.org) выдает данные о погоде в двух форматах: xml и json. Напишите программу, которая будет принимать файл с данными о погоде и извлекать из него необходимые данные.

**Задача 1**. Откройте Git CMD или Git Bash и выполните настройку пользователя, от которого будет идти работа. Используя следующие команды, введите свои имя и почту:

git config --global user.name ”Ivan Ivanov”

git config --global user.email ivan.ivanov@gmail.com

**Задача 2**. Создайте проект в VS c именем **WeatherService**, добавьте его под контроль Git – инициализируйте новый репозиторий:

git init

Переименуйте ветку master в main:

git branch -m main

**Задача 3**. С помощью команды

git status

определите файлы, которые нужно внести под версионный контроль. Подсказка: это файлы, названия которых начинаются с WeatherService (возможно, некоторые будут в папке WeatherService). Добавьте их под версионный контроль:

git add имя\_файла

git add имя\_папки

**Задача 4**.

Можно было добавить файлы привычной командой

git add .

Но в этом случае будут добавлены и скрытые файлы, которые создает сама VS. Нужно "пометить" их как игнорируемые.

Для этого добавьте в репозиторий файл **.gitignore**. Обратите внимание, этот файл должен быть в том же каталоге, в котором создалась скрытая папка **.git**.

С помощью текстового редактора добавьте в файл .gitignore скрытую папку **.vs**. Далее, нужно будет добавить папку Debug (она создается при запуске проекта). Подробнее про то, как заполнить файл .gitignore <https://ru.hexlet.io/courses/intro_to_git/lessons/gitignore/theory_unit>, <https://www.atlassian.com/ru/git/tutorials/saving-changes/gitignore>.

Добавьте под версионный контроль файл .gitignore. Выполните коммит.

**Задача 5**. Создайте класс Weather для хранения параметров погоды. Проанализировав оба файла (json и xml), можно выделить следующие общие данные: название населенного пункта и его координаты (широта и долгота), значение температуры, описание погоды, скорость ветра и облачность. Добавьте в класс следующие поля:

std::string city; // Киров

double lon; // 49.6601

double lat; // 58.5966

double temperature; // 5.69

std::string weather; // дождь

double windSpeed; // 4.27

int clouds; // 100

Опишите конструктор со всеми параметрами. Добавьте возможность вывода описания погоды (не обязательно это делать сейчас, возможно позже).

Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 6**. Создайте новую ветку dev от ветки main

git branch dev

перейдите в нее

git checkout dev

Создайте новый класс Service (для него достаточно только h-файла). Добавьте в открытую часть класса чистый виртуальный метод:

virtual Weather getWeather(std::string s) = 0;

и виртуальный деструктор.

Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 7**. Выполните слияние ветки dev в ветку main. Для этого сначала перейдите в ветку main и выполняйте команды оттуда

git merge dev

Подробнее про особенности слияния <https://www.atlassian.com/ru/git/tutorials/using-branches/git-merge>.

**Задача 8**. Создайте новую ветку feature-json от ветки main, перейдите в нее. Создайте класс с именем JsonService, унаследуйте его от класса Service.

class JsonService :public Service

{

public:

virtual Weather getWeather(std::string s) override;

virtual ~JsonService() {};

};

Реализация метода getWeather. Добавьте к проекту необходимую библиотеку. Вместо многоточия впишите верный код.

using nlohmann::json;

Weather JsonService::getWeather(std::string s)

{

std::ifstream fin(s);

if (!fin)

throw exception("error");

json j;

j = json::parse(fin);

std::string city = j["name"]; // Киров

double lon = j["coord"]["lon"]; // 49.6601

double lat = ...; // 58.5966

double temperature = ...; // 5.69

std::string weather = ...; // дождь

double windSpeed = ...; // 4.27

int clouds = j["clouds"]["all"]; // 100

return Weather(параметры);

}

Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 9**. В функции main проверьте, правильно ли создается объект класса погоды из созданного сервиса.

JsonService js;

Weather w = js.getWeather("weather.json");

При запуске проекта создается папка Debug, добавьте ее в .gitignore. Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 10**. Перейдите на ветку dev. Выполните слияние ветки feature-json в ветку dev.

**Задача 11**. Создайте новую ветку feature-xml от ветки main, перейдите в нее. Если все сделано правильно, то код не должен содержать файлов JsonService.h и JsonService.cpp.

Создайте класс с именем XmlService, унаследуйте его от класса Service.

class XmlService:public Service

{

public:

virtual Weather getWeather(std::string s) override;

virtual ~XmlService() {};

};

Добавьте к проекту необходимую библиотеку. Вместо многоточия впишите верный код.

using namespace pugi;

Weather XmlService::getWeather(std::string s)

{

xml\_document doc;

xml\_parse\_result result = doc.load\_file(s.c\_str());

if (!result)

throw exception("error");

xml\_node node = doc.child("current");

std::string city = node.child("city").

attribute("name").

as\_string(); // Киров

double lon = node.child("city").child("coord").

attribute("lon").

as\_int(); // 49.6601

double lat = ...; // 58.5966

double temperature = node.child("temperature").

attribute("value").

as\_double(); // 5.69

std::string weather = ...; // дождь

double windSpeed = ...; // 4.27

int clouds = ...; // 100

return Weather(параметры);

}

Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 12**. В функции main проверьте, правильно ли создается объект класса погоды из созданного сервиса.

XmlService xs;

Weather w = xs.getWeather("weather.xml");

Сохраните изменения. Проиндексируйте новые файлы, выполните коммит.

**Задача 13**. Перейдите на ветку dev. Выполните слияние ветки feature-xml в ветку dev.

Разрешите конфликт в функции main – оставьте создание погоды двумя способами. После редактирования (и сохранения изменений) файла git советует выполнить коммит, сделайте это.

**Задача 14**. Перейдите на ветку main. Выполните слияние ветки dev в ветку main.

При возникновении конфликта, разрешите его.

**Задача 15**. Создайте удаленный репозиторий и выполните push (со всеми ветками) в него. Для этого может понадобится подключиться к удаленному репозиторию:

git add remote origin url(удаленного репозитория)

Чтобы "запушить" все ветки нужно добавить ключ --all (иначе добавится только main):

git push origin --all

**Задача 16**. В moodle скиньте ссылку на свой репозиторий на GitHub.

Полезная ссылка: <https://docs.github.com/ru/get-started>