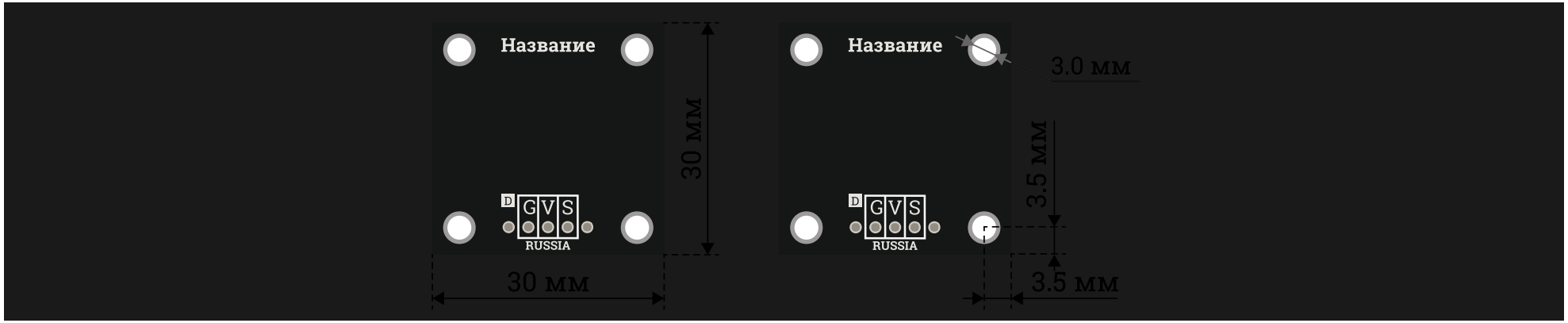
Подключение и работа с флеш картой на Arduino

Общие сведения:

Trema-модуль адаптер microSD — адаптер карт памяти microSD, который вы можете использовать для записи значений аналоговых и цифровых выводов Arduino.

Спецификация:

- Поддерживаемые карты памяти: microSD-карты (FAT16/FAT32);
- Объём карты памяти: SD до 2Gb / SDHC до 32Gb;
- Питание: 5В;
- Интерфейс: SPI;
- Вывод INS для идентификации наличия вставленной карты
- Все модули линейки "Trema" выполнены в одном формате



Подключение:

Для удобства подключения к Arduino воспользуйтесь Trema Shield, Trema Power Shield, Motor Shield или Trema Set Shield.

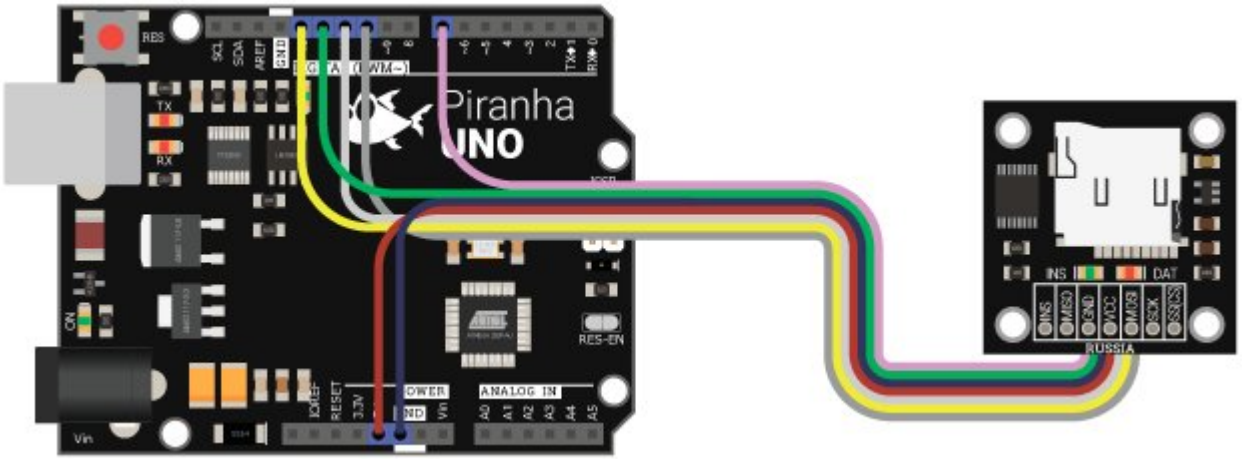
Выводы Trema SD	Выводы Piranha Uno/Arduino Uno	Выводы Arduino Mega	Выводы Arduino Leonardo

MISO	12	50	ICSP_MISO
MOSI	11	51	ICSP_MOSI
SCK	13	52	ICSP_SCK
SS(SC)	10	53	ICSP_SS
INS	Любой вывод	Любой вывод	Любой вывод
GND	GND	GND	GND
Vcc	5V	5V	5V

Модуль удобно подключать 5 способами, в зависимости от ситуации:

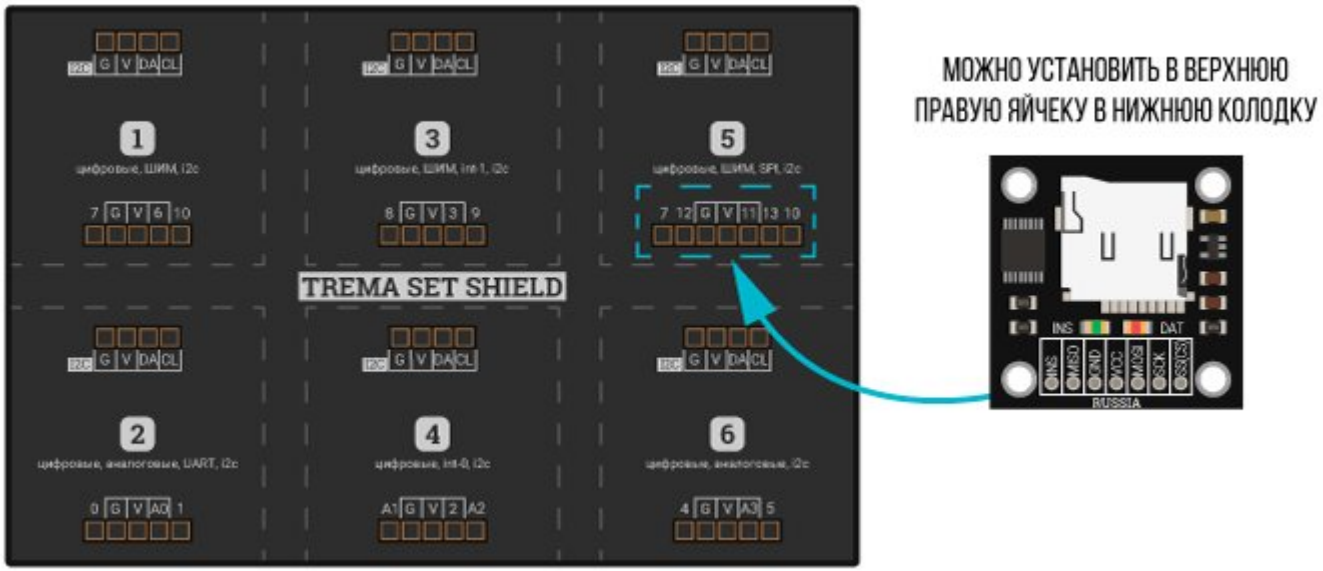
Способ - 1 : Используя проводной шлейф и Piranha UNO

Используя провода «Папа — Мама», подключаем напрямую к контроллеру Piranha UNO



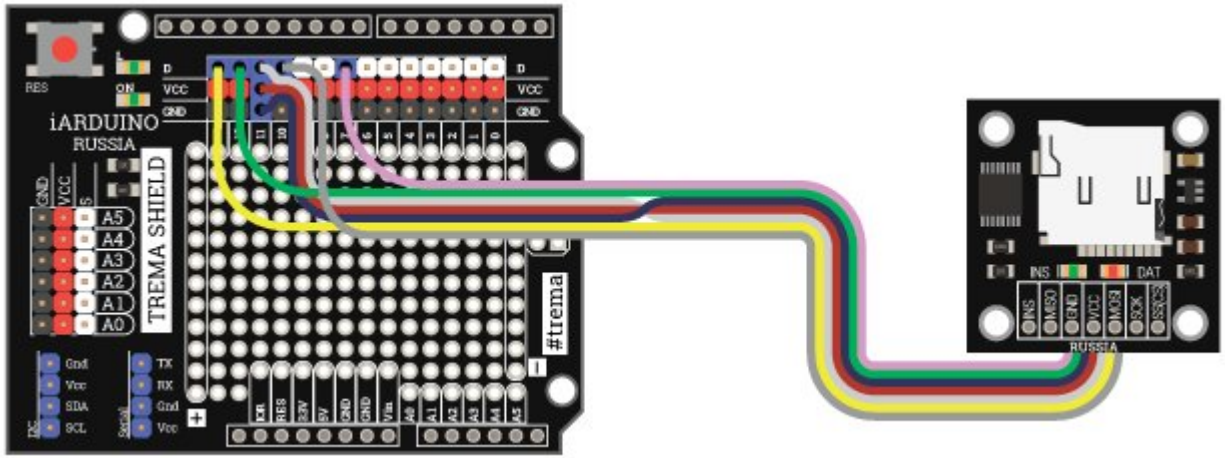
Способ - 2 : Используя Trema Set Shield

Модуль можно подключить к любому из входов Trema Set Shield (при использовании программного последовательного порта)



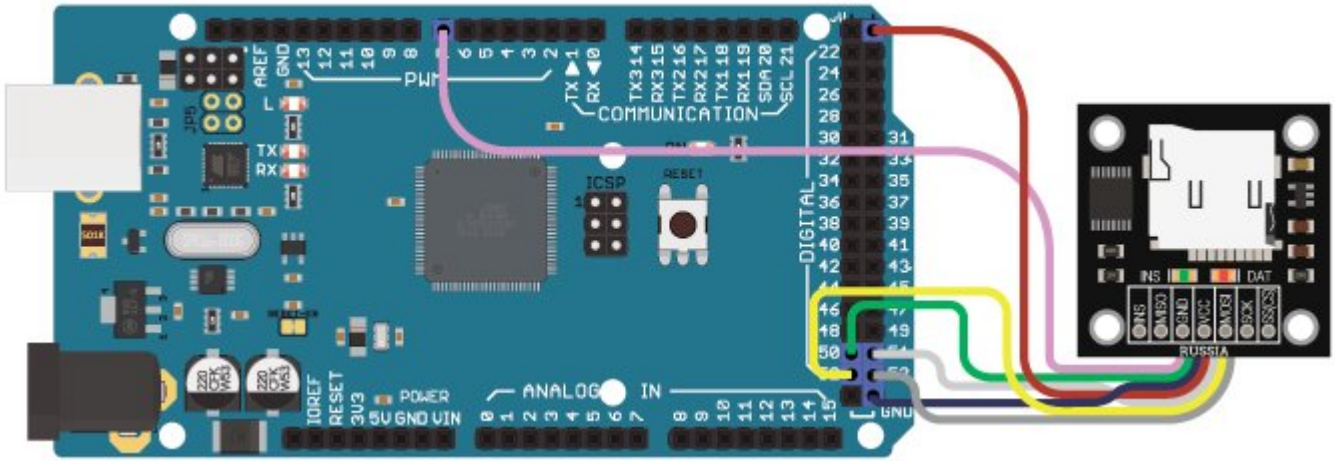
Способ - 3 : Используя проводной шлейф и Trema Shield

Используя 3-х проводной шлейф и 4 провода «Мама — Мама», к Trema Shield, Trema-Power Shield, Motor Shield, Trema Shield NANO и тд.



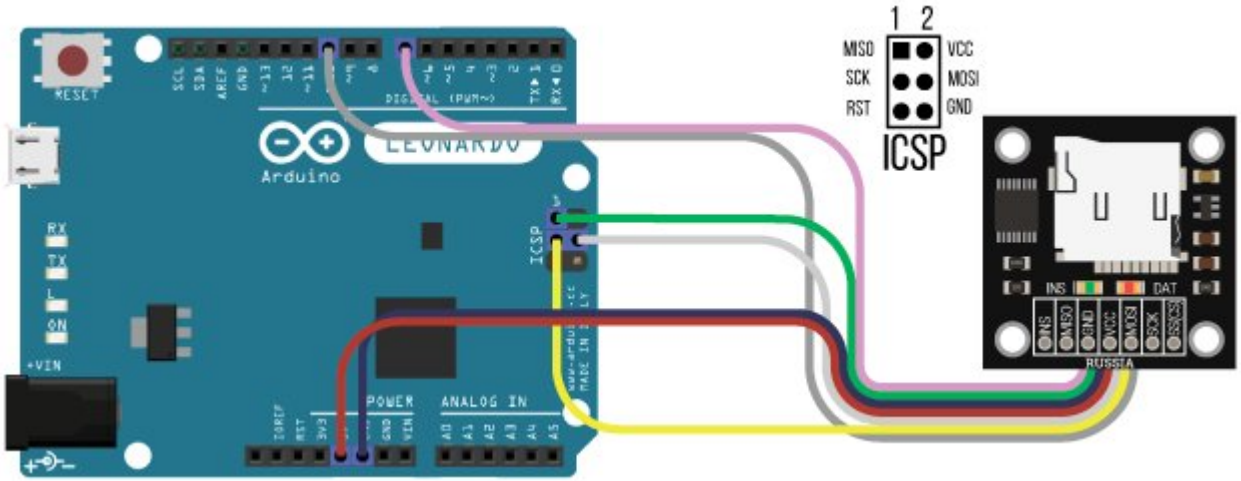
Способ - 4 : Используя провода и Arduino Mega

Используя провода «Папа — Мама», подключаем напрямую к контроллеру Arduino Mega



Способ - 5 : Используя провода и Arduino Leonardo

Используя 4 провода «Папа — Мама» и 3 провода «Мама — Мама», подключаем напрямую к контроллеру Arduino Leonardo



Питание:

Данный модуль питается от источника постоянного напряжения 5В.

Подробнее о модуле:

Модуль поддерживает карты microSD. Для работы с библиотекой Arduino карта должна быть отформатирована в формате FAT16 или FAT32.

У модуля есть возможность снизить энергопотребление путём отключения индикаторных светодиодов. Это производится посредством удаления двух напаянных джамперов на нижней стороне платы:

На модуле имеется вывод "**INS**", определяющий наличие карты.

Примеры:

Внимание: во избежание потери данных рекомендуем извлекать microSD карту во время загрузки скетча в микроконтроллер.

Инициализация карты, создание и запись файла

```
#include <SPI.h>                                          // Подключаем библиотеку для работы с шиной SPI
#include <SD.h>                                           // Подключаем библиотеку для работы с SD картой

#define INS 7                                             // определяем номер вывода, отвечающего за определение наличия к

void setup() {

  Serial.begin(9600);                                     // инициализируем работу с аппаратным последовательным портом и
  while(!Serial);                                         // Ждём соединения с последовательном портом (необходимо для сов
  pinMode(INS, INPUT_PULLUP);                             // объявляем режим вывода INS
  Serial.println("Waiting for the card to be inserted");  // выводим текст в последовательный порт.
  while(digitalRead(INS));                                // ждем пока будет вставлена карта
  delay(1000);                                            // ждем секунду, чтобы убедиться что карта вставлена полностью
  if (!SD.begin()) {                                      // Проверяем, есть ли связь с картой и, если нет, то
    Serial.println("Initialization failed");              // выводим текст в последовательный порт.
    while(true);                                          // Скетч не выполняется дальше
  }
  Serial.println("Card OK");                              // выводим текст в последовательный порт.
  File myFile = SD.open("file.txt", FILE_WRITE);          // создаём файл для записи
  char c = '@';                                           // создаём переменную
  int i = 42;                                             // создаём переменную
  float f = 3.14;                                         // создаём переменную
  String s = "trema";                                     // создаём переменную
  myFile.println(c);                                      // записываем переменную в файл
  myFile.println(i);                                      // записываем переменную в файл
  myFile.println(f);                                      // записываем переменную в файл
  myFile.println(s);                                      // записываем переменную в файл
  myFile.close();                                         // закрываем файл
  Serial.println("done");                                 // выводим текст в последовательный порт.
}
```

```
void loop() {
 //
}
```

Чтение из файла в последовательный порт

```
#include <SPI.h>                      // Подключаем библиотеку для работы с протоколом SPI
#include <SD.h>                       // Подключаем библиотеку для работы с SD картой

void setup()
{
  Serial.begin(9600);                 // инициализируем работу с аппаратным последовательным портом и указываем скорость рабо
  while(!Serial);                     // Ждём соединения с последовательным портом (необходимо для совместимости с Arduino Le
  if (!SD.begin()) {                  // Проверяем, есть ли связь с картой и, если нет, то
    Serial.println("Initialization\
                    failed");         // выводим текст в последовательный порт.
    while(true);                      // Скетч не выполняется дальше.
  }
  Serial.println("Card OK");          // Выводим текст в последовательный порт.
  File myFile = SD.open("file.txt");  // Открываем файл для чтения,
  while(myFile.available())
        Serial.write(myFile.read());  // выводим содержимое файла в последовательный порт,
  myFile.close();                     // закрываем файл
}

void loop()
{
  //
}
```

Чтение из файла в переменные и удаление файла

```
#include <SPI.h>                                  // Подключаем библиотеку для работы с протоколом SPI
#include <SD.h>                                   // Подключаем библиотеку для работы с SD картой

char c = 0;                                       // создаём глобальную переменную для чтения из файла
int i = 0;                                        // создаём глобальную переменную для чтения из файла
float f = 0;                                      // создаём глобальную переменную для чтения из файла
String s = "";                                    // создаём глобальную переменную для чтения из файла

void setup() {

  Serial.begin(9600);                             // инициализируем работу с аппаратным последовательным портом и указываем ск
  while(!Serial);                                 // Ждём соединения с последовательном портом (необходимо для совместимости с
  if (!SD.begin()) {                              // проверяем, есть ли связь с картой и, если нет, то
    Serial.println("Initialization failed");      // выводим текст в монитор последовательного порта,
    while(true);                                  // скетч не выполняется дальше.
  }
  Serial.println("Card OK");                      // выводим текст в монитор последовательного порта,
  File myFile = SD.open("file.txt");              // открываем ранее созданный файл file.txt

  if (!myFile) {                                  // если файл file.txt не найден
    Serial.println("file not found");             // выводим текст в монитор последовательного порта,
  while(true);                                    // скетч не выполняется дальше.

  } else {                                        // Если файл file.txt найден,

  char Bytes[10];                                 // создаём переменную для хранения байтов в формате char,
  int k = 0;                                      // создаём переменную-итератор,

  while(myFile.available() &&                     // пока файл не кончился,
    ((Bytes[k] = myFile.read()) != '\n')) ++k;    // читаем байты из файла до символа новой строки,
```

```
    c = Bytes[0];                                // записываем первый байт в переменную char,
    k = 0;                                       // сбрасываем итератор,

  while(myFile.available() &&                    // пока файл не кончился,
    ((Bytes[k] = myFile.read()) != '\n')) ++k;   // читаем байты дальше до следующего символа новой строки,

    i = atoi(Bytes);                             // конвертируем в int,
    k = 0;                                       // сбрасываем итератор,

  while(myFile.available() &&                    // пока файл не кончился,
    ((Bytes[k] = myFile.read()) != '\n')) ++k;   // читаем байты дальше до следующего символа новой строки,

    f = atof(Bytes);                             // конвертируем в float,

  while(myFile.available() &&                    // пока файл не кончился,
    ((s += char(myFile.read()))));               // читаем байты и сразу конкатенируем в переменную String.

  myFile.close();                                // Закрываем файл.

  SD.remove("file.txt");                         // Удаляем файл.

  }
}

void loop() {

  Serial.println(c);                             // Выводим переменную в последовательный порт.
  Serial.println(i);                             // Выводим переменную в последовательный порт.
  Serial.println(f);                             // Выводим переменную в последовательный порт.
  Serial.println(s);                             // Выводим переменную в последовательный порт.
  delay(i*100);                                  // Ждём 4200 миллисекунд
```

```
}
```

Информация о карте и листинг файлов, используя утилитные функции библиотеки SD

```
#include <SPI.h>                          // Подключаем библиотеку для работы с протоколом SPI
#include <SD.h>                           // Подключаем библиотеку для работы с SD картой


Sd2Card card;                             // создаём объект утилитной библеотеки Sd2Card
SdVolume partition;                       // создаём объект тома утилитной библеотеки Sd2Card
SdFile root;                              // создаём объект корневого файла утилитной библеотеки Sd2Card

void setup()
{
  Serial.begin(9600);                     // инициализируем работу с аппаратным последовательным портом,
  while (!Serial);                        // ждём соединения с последовательным портом (необходимо для совместимости с Ardui

  if (!card.init(SPI_HALF_SPEED)){        // Если карта не найдена (SPI_HALF_SPEED = F_CPU/4),
    Serial.println("initialization\
                    failed");             // выводим текст в монитор последовательного порта.
    while(true);                          // Скетч не выполняется дальше.
  }
  else
    Serial.println("Card OK");            // Если карта найдена

  switch (card.type()){                   // проверяем тип карты:
    case SD_CARD_TYPE_SD1:                // тип карты SD1,
      Serial.println("SD1");
      break;
    case SD_CARD_TYPE_SD2:
      Serial.println("SD2");              // тип карты SD2,
      break;
```

```
    case SD_CARD_TYPE_SDHC:
      Serial.println("SDHC");                 // тип карты SDHC,
      break;
    default:
      Serial.println("Unknown type");         // тип карты неизвестен.
  }

  if (!partition.init(card)) {                // Открываем раздел карты, если не найден -
    Serial.println("No partition found.\      // выводим текст в монитор последовательного порта.
                    Make sure card is\
                    is formatted as\
                    FAT16 or FAT32");
    while(true);                              // Скетч не выполняется дальше.
  }

  uint32_t part_size;                         // Создаём переменную для расчета размера раздела карты:
  part_size = partition.blocksPerCluster(); // кол-во блоков в кластере
  part_size *= partition.clusterCount();    // умножаем на кол-во кластеров,
  part_size /= 2;                           // 1 блок - 512 байтов.
  part_size /= 1024;                        // Размер в мегабайтах.
  Serial.print("Partition size in Mb: ");   // Выводим текст в монитор последовательного порта.
  Serial.println(part_size);                // Выводим размер в мегабайтах в монитор последовательного порта.
  Serial.print("Partition size in Gb: ");   // Выводим текст в монитор последовательного порта.
  Serial.print(float(part_size) / 1024.0);  // Выводим размер в гигабайтах в монитор последовательного порта.
  Serial.println("Root directory:");        // Выводим текст в монитор последовательного порта.
  root.openRoot(partition);                 // Открываем корневой каталог раздела,
  root.ls (LS_R | LS_SIZE, 3);              // вызываем утилитную функцию листинга открытого каталога.

}


void loop()
{
```

```
//
}
```

Листинг файлов без использования утилитных функций

```
#include <SPI.h>                                // Подключаем библиотеку для работы с протоколом SPI
#include <SD.h>                                 // Подключаем библиотеку для работы с SD картой

void setup() {

  Serial.begin(9600);                           // инициализируем работу с аппаратным последовательным портом и указываем скор
  while(!Serial);                               // Ждём соединения с последовательном портом (необходимо для совместимости с A
  if (!SD.begin()) {                            // Проверяем, есть ли связь с картой и, если нет, то
    Serial.println("Initialization failed");     // выводим текст в монитор последовательного порта.
    while(true);                                // Скетч не выполняется дальше
  }
  Serial.println("Card OK");                    // выводим текст в монитор последовательного порта.
  Serial.println("Type: ");                     // выводим текст в монитор последовательного порта.

  File root = SD.open("/");                     // открываем корневой каталог
  printRoot(root);                              // вызываем функцию листинга
}

void loop() {
//
}

void printRoot(File dir) {                      // функция листинга директории.
   while(true) {
     File entry =  dir.openNextFile();          // Открываем следующий файл,
```

```
    if (! entry) {                          // если файлов не осталось
      break;                                // выходим из цикла.
    }
    Serial.print(entry.name());             // Выводим имя файла в последовательный порт.
    if (entry.isDirectory()) {              // Если файл является директорией
      Serial.println("/");                  // Выводим косую черту,
      printRoot(entry);                     // рекурсия в директорию (функция вызывает сама себя).
    } else {                                // Если не директория -
      Serial.print("\t\t\t\t");             // выводим отступ
      Serial.println(entry.size(), DEC);    // и размер файла в байтах
    }
  }
}
```

Создание лог-файла с использованием часов реального времени и датчика звука.

```
#include "SPI.h"                                        // подключаем библиотеку для работы с шиной SPI
#include "SD.h"                                         // подключаем библиотеку для работы с SD картой
#include "iarduino_RTC.h"                               // подключаем библиотеку

iarduino_RTC time(RTC_DS1307);                          // объявляем объект time для модуля на базе чипа DS1307
File myFile;                                            // объявляем объект myFile для работы с файлом на SD-карте

const uint8_t SOUND_SENSOR = A1;                        // вывод, к которому подключен датчик звука
String        REAL_TIME;                                // переменная для значений реального времени
uint16_t      LOUDNESS;                                 // переменная для аналоговых значений датчика звука

void setup() {
  Serial.begin(9600);                                   // инициализируем работу с аппаратным последовательным портом и
  pinMode(SOUND_SENSOR, INPUT);                         // настраиваем вывод A1 на работу в режиме ВХОДА
  time.begin();                                         // Инициируем RTC модуль
  time.settime(0, 15, 15, 14, 5, 19, 2);                // Устанавливаем время: 0 сек, 15 мин, 15 час, 14, мая, 2019 год
```

```
  if (!SD.begin()) {                                    // Проверяем, есть ли связь с картой и, если нет, то
    Serial.println("Initialization failed");            // выводим текст в последовательный порт.
    while (true);                                       // Скетч не выполняется дальше
  }
  Serial.println("Card OK");                            // выводим текст, что всё прошло успешно, в последовательный пор
}

void loop() {
  if (millis() % 5000 == 0) {                           // если прошло 5 секунд, то
    REAL_TIME = time.gettime("d-m-Y, H:i:s");           // записываем значение, считанное с часов реального времени, в г
    LOUDNESS = analogRead(SOUND_SENSOR);                // записываем значение, считанное с датчика звука, в переменную
    myFile = SD.open("test.txt", FILE_WRITE);           // создаём файл test.txt для записи значений
    myFile.print(REAL_TIME);                            // записываем переменную в файл
    myFile.print(",");                                  // записываем разделительный символ
    myFile.println(LOUDNESS);                           // записываем переменную в файл
    myFile.close();                                     // закрываем файл
  }
}
```

Описание функций библиотеки:

На заметку: Библиотека SD входит в стандартный набор библиотек Arduino

Подключение библиотеки:

```
#include <SPI.h>                                        // Подключаем библиотеку для работы с шиной SPI
#include <SD.h>                                         // Подключаем библиотеку для работы с SD картой
```

Функция begin();

- Назначение: инициирование работы с картой;
- Синтаксис: **begin();**
- Параметры: ВЫВОД CS(chip select) – опциональный параметр, по умолчанию вывод SS(slave select) шины SPI. Вывод 10 на Piranha Uno, Arduino Leonardo; вывод 53 на Arduino Mega;
- Возвращаемые значения: bool - результат инициализации (true / false);
- Примечание:
  - Функцию необходимо вызвать до обращения к любым другим функциям библиотеки;
  - Функцию достаточно вызвать один раз в коде `setup` ;
  - Функцию можно использовать для определения наличия SD карты;
- Пример:

```
SD.begin(); //Инициализируем SD карту
```

Функция exists();

- Назначение: проверка существования файла или директории на SD карте;
- Синтаксис: **exists(ИМЯ)**;
- Параметры: ИМЯ — строка String, имя файла для проверки. Может включать директории, разделённые косой чертой (прямой слэш - /);
- Возвращаемые значения: bool — **true** - файл существует/ **false** - файл не существует;
- Примечание: нет;
- Пример:

```
SD.exists("file.txt"); // проверяем наличие файла "file.txt"
```

Функция mkdir();

- Назначение: проверка существования файла или директории на SD карте;
- Синтаксис: **mkdir(ИМЯ)**;
- Параметры: ИМЯ – строка String, имя создаваемой директории
- Возвращаемые значения:

- **true** - директория создана / **false** – директория не создана;
- Примечание: нет;
- Пример:

```
SD.mkdir("FOLDER1/FOLDER2/FOLDER3"); // создаём папку
```

Функция open();

- Назначение: открытие файла, создание файла (файл будет создан, если открыт в режиме записи и не существует);
- Синтаксис: **open(ИМЯ), open(ИМЯ, РЕЖИМ)**;
- Параметры: **ИМЯ** — строка **String**, имя файла. Может включать директории, разделённые косой чертой (прямой слэш - /). **РЕЖИМ** – целое число без знака, **uint_8t**: **FILE_READ** – открыть для чтения, **FILE_WRITE** – открыть для записи). По умолчанию **FILE_READ**;
- Возвращаемые значения:
- объект **File**. Если файл не найден, объект возвращает **false**;
- Примечание: нет;
- Пример:

```
SD.open("file.txt", FILE_WRITE); // открываем файл file.txt для записи
```

Функция remove();

- Назначение: удаление файла.;
- Синтаксис: **remove(ИМЯ)**;
- Параметры: **ИМЯ** — имя файла для удаления;
- Возвращаемые значения:
- при успешном выполнении **true**, иначе **false**;
- Примечание: нет;
- Пример:

```
SD.remove("file.txt"); // удаляем файл file.txt
```

Функция rmdir();

- Назначение: удаление пустой директории (папки);
- Синтаксис: **rmdir(ИМЯ)**;
- Параметры: **ИМЯ** — имя директории (папки) для удаления;
- Возвращаемые значения:
- при успешном выполнении **true**, иначе **false**;
- Примечание: нет;
- Пример:

```
SD.rmdir("FOLDER"); // удаляем папку FOLDER
```

Описание объекта File:

Инициализация объекта:

```
Files myFile=SD.open("file.txt"); // открываем файл для чтения
```

Функция name();

- Назначение: возврат имени открытого файла в объекте myFile;
- Синтаксис: **name()**;
- Параметры: нет
- Возвращаемые значения:
- строка **String** – имя файла;
- Примечание: нет;
- Пример:

```
String filename = myFile.name(); // создаём переменную строки, хранящую название файла
```

Функция available();

- Назначение: возврат количества байтов, доступных для чтения;
- Синтаксис: **available()**;
- Параметры: нет
- Возвращаемые значения:
- целое число **int** — количество байтов для чтения;
- Примечание: нет;
- Пример:

```
int bytes = myFile.available(); // записываем количество доступных байтов в целочисленную переменную bytes
```

Функция close();

- Назначение: закрытие файла и запись данных;
- Синтаксис: **close()**;
- Параметры: нет;
- Возвращаемые значения: нет;
- Примечание: нет;
- Пример:

```
myFile.close(); // закрываем открытый файл
```

Функция flush();

- Назначение: записать требующие записи данные на карту, выполняется автоматически при использовании функции **close()**;
- Синтаксис: **exists()**;
- Параметры: нет;
- Возвращаемые значения: нет;
- Примечание: нет;
- Пример:

```
myFile.flush(); // записываем несохраненные данные на SD карту
```

Функция peek();

- Назначение: Чтение байта из файла без сдвига позиции чтения. При следующем вызове функции будет возвращено то же значение;
- Синтаксис: **peek()**;
- Параметры: нет
- Возвращаемые значения:
- **byte** или -1 (**EOF**), если байтов для чтения не осталось;
- Примечание: нет;
- Пример:

```
myFile.peek(); // считываем текущий байт
```

Функция position();

- Назначение: возврат позиции в файле, в которую будет записан следующий байт;
- Синтаксис: **position()**;
- Параметры: нет
- Возвращаемые значения:
- **unsigned long** — значение позиции;
- Примечание: нет;
- Пример:

```
myFile.position(); // узнаём позицию в файле
```

Функция print();

- Назначение: проверка существования файла или директории на SD карте;

- Синтаксис: **print(ДАННЫЕ); print(ДАННЫЕ, СИСТЕМА СЧИСЛЕНИЯ)**;
- Параметры: **ДАННЫЕ** — данные для записи **char**, **int**, **long**, **byte**, **String**; **СИСТЕМА СЧИСЛЕНИЯ** — опционально (по умолчанию DEC), система счисления в которой будут записаны числа: BIN - двоичная, DEC - десятеричная, OCT - восьмеричная, HEX — шестнадцатеричная;
- Возвращаемые значения:
- **byte** — количество записанных байтов;
- Примечание: нет;
- Пример:

```
int x = 123;
myFile.print(x, HEX); // записываем в файл значение переменной в шестнадцатеричной системе счисления (будет записано "7B")
```

Функция println();

- Назначение: запись данных в файл, открытый для записи с последующим символом новой строки;
- Синтаксис: **println()**; **println(ДАННЫЕ)**; **println(ДАННЫЕ, СИСТЕМА СЧИСЛЕНИЯ)**;
- Параметры: **ДАННЫЕ** — данные для записи **char**, **int**, **long**, **byte**, **String**; **СИСТЕМА СЧИСЛЕНИЯ** — опционально (по умолчанию DEC), система счисления в которой будут записаны числа: BIN - двоичная, DEC - десятеричная, OCT - восьмеричная, HEX — шестнадцатеричная;
- Возвращаемые значения:
- количество записанных байтов;
- Примечание: нет;
- Пример:

```
myFile.println(); // записываем в открытый файл символ новой строки
```

Функция seek();

- Назначение: перемещение на новую позицию в файле;
- Синтаксис: **seek(ПОЗИЦИЯ)**;
- Параметры: ПОЗИЦИЯ — **unsigned long**, позиция в файле на которую нужно переместиться;
- Возвращаемые значения:

- **true** - успешное выполнение / **false** - неуспешное выполнение;
- Примечание: нет;
- Пример:

```
bool out = myFile.seek(myFile.size()+1); // перемещение на несуществующую позицию в файле.
```

Функция size();

- Назначение: Узнать размер открытого файла;
- Синтаксис: **size()**;
- Параметры: нет
- Возвращаемые значения:
- размер файла в **unsigned long**;
- Примечание: нет;
- Пример:

```
unsigned long filesize = myFile.size(); // узнаём размер открытого файла
```

Функция read();

- Назначение: проверка существования файла или директории на SD карте;
- Синтаксис: **read(БУФЕР, ДЛИННА)**;
- Параметры: **БУФЕР** — массив элементов, **ДЛИННА** — количество элементов в массиве;
- Возвращаемые значения:
- **byte** следующий байт / **EOF** конец файла;
- Примечание: нет;
- Пример:

```
byte data = myFile.read(); // записываем прочитанный байт в переменную data
```

Функция write();

- Назначение: запись в файл без конвертирования;
- Синтаксис: **write(ДАННЫЕ), write(БУФЕР, ДЛИННА)**;
- Параметры: **ДАННЫЕ** - данные для записи byte, char или String. БУФЕР — массив символов или байтов для записи, **ДЛИННА** — количество элементов массива
- Возвращаемые значения:
- количество записанных байт;
- Примечание: нет;
- Пример:

```
myFile.write(123); // записываем в виде байта число 123
```

Функция isDirectory();

- Назначение: проверка существования файла или директории на SD карте;
- Синтаксис: **isDirectory()**;
- Параметры: нет
- Возвращаемые значения:
- **true** - файл является директорией / **false** - файл не является директорией;
- Примечание: нет;
- Пример:

```
if (myFile.isDirectory()) Serial.println("/"); //выводим слэш, если файл является директорией
```

Функция openNextFile();

- Назначение: возвращает следующий файл в директории;
- Синтаксис: **openNextFile()**;
- Параметры: нет
- Возвращаемые значения: объект **File** или **false**, если файл не может быть открыт;

- следующий файл или папка в открытой директории;
- Примечание: нет;
- Пример:

```
File dir = SD.open("/");              //открываем корневую директорию SD карты
File myFile = dir.openNextFile();     //открываем первый файл в директории
```

Функция rewindDirectory();

- Назначение: проверка существования файла или директории на SD карте;
- Синтаксис: **rewindDirectory()**;
- Параметры: нет;
- Возвращаемые значения:нет;
- Примечание: нет;
- Пример:

```
if (!myFile) dir.rewindDirectory(); //возвращаемся к первому файлу, если файл не существует
```