



**Министерство науки и высшего образования Российской  
Федерации  
Федеральное государственное бюджетное образовательное  
учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)**

---

ФАКУЛЬТЕТ	Информатика и системы управления
КАФЕДРА	Информационная безопасность (ИУ8)

Отчёт  
по лабораторной работе №3  
по дисциплине «Безопасность систем баз данных»

Выполнил: Песоцкий А. А.,  
студент группы ИУ8-61

Проверил: Зенькович С. А.,  
ассистент каф. ИУ8

Москва 2020 г.

# ОГЛАВЛЕНИЕ

## Table of Contents

<u>ВСТУПЛЕНИЕ.....</u>	<u>3</u>
<u>ЧТО ТАКОЕ POSIX.....</u>	<u>4</u>
<u>ПРИМЕР ИСПОЛЬЗОВАНИЯ SHELL-СКРИПТА.....</u>	<u>5</u>
<u>ПРИМЕР ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ .....</u>	<u>8</u>
<u>ЗАКЛЮЧЕНИЕ .....</u>	<u>10</u>
<u>ПРИЛОЖЕНИЕ.....</u>	<u>11</u>

## ВСТУПЛЕНИЕ

### Цель работы:

Разработать приложение, которое должно уметь:

1. Запускать произвольное приложение.
2. Получать его поток stdout, и:
  - записывать в файл;
  - выводить в консоль.
3. Получать его поток stderr, и:
  - записывать в файл;
  - выводить в консоль.
4. Получать его код завершения.

При этом каждая “опция” п. 2-3 должна быть настраиваемой. Вывод информации из п. 4 должен происходить в консоль, и, если есть открытые на запись файлы - дублироваться в них. Приложение должно быть реализовано в 2-х вариантах:

1. POSIX-совместимый shell скрипт.
2. Приложение на C99/C++11.

# ЧТО ТАКОЕ POSIX

**POSIX** (англ. *Portable Operating System Interface* — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносимости прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

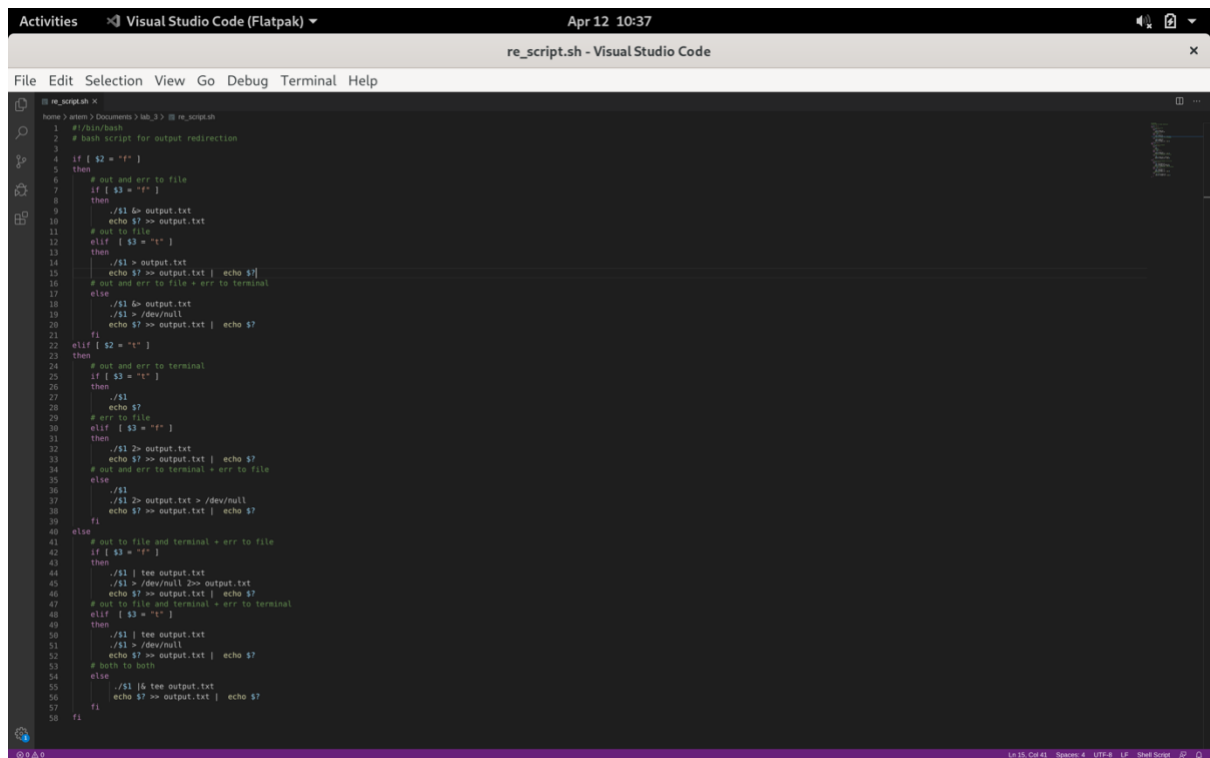
Стандарт состоит из четырёх основных разделов.

- **Основные определения** (англ. *Base definitions*) — список основных определений и соглашений, используемых в спецификациях, и список заголовочных файлов языка Си, которые должны быть предоставлены соответствующей стандарту системой.
- **Оболочка и утилиты** (англ. *Shell and utilities*) — описание утилит и командной оболочки sh, стандарты регулярных выражений.
- **Системные интерфейсы** (англ. *System interfaces*) — список системных вызовов языка Си.
- **Обоснование** (англ. *Rationale*) — объяснение принципов, используемых в стандарте.

Задачи POSIX:

- Содействовать облегчению переноса кода прикладных программ на иные платформы.
- Способствовать определению и унификации интерфейсов заранее при проектировании, а не в процессе их реализации.
- Сохранять по возможности и учитывать все главные, созданные ранее и используемые прикладные программы.
- Определять необходимый минимум интерфейсов прикладных программ для ускорения создания, одобрения и утверждения документов.
- Развивать стандарты в направлении обеспечения коммуникационных сетей, распределенной обработки данных и защиты информации.
- Рекомендовать ограничение использования бинарного (объектного) кода для приложений в простых системах.

# ПРИМЕР ИСПОЛЬЗОВАНИЯ SHELL-СКРИПТА



```
#!/bin/bash
# bash script for output redirection
1
2
3
4 if [ $2 = "f" ]
5 then
6     # out and err to file
7     if [ $3 = "f" ]
8     then
9         ./s1 &> output.txt
10        echo $7 >> output.txt
11        # out to file
12        elif [ $3 = "t" ]
13        then
14            ./s1 > output.txt
15            echo $7 >> output.txt | echo $7
16            # out and err to file + err to terminal
17        else
18            ./s1 &> output.txt
19            ./s1 > /dev/null
20            echo $7 >> output.txt | echo $7
21        fi
22    elif [ $2 = "t" ]
23    then
24        # out and err to terminal
25        if [ $3 = "f" ]
26        then
27            ./s1
28            echo $7
29            # err to file
30            elif [ $3 = "f" ]
31            then
32                ./s1 2> output.txt
33                echo $7 >> output.txt | echo $7
34            # out and err to terminal + err to file
35        else
36            ./s1
37            ./s1 2> output.txt > /dev/null
38            echo $7 >> output.txt | echo $7
39        fi
40    else
41        # out to file and terminal + err to file
42        if [ $3 = "f" ]
43        then
44            ./s1 | tee output.txt
45            ./s1 > /dev/null 2>> output.txt
46            echo $7 >> output.txt | echo $7
47            # out to file and terminal + err to terminal
48        elif [ $3 = "t" ]
49        then
50            ./s1 | tee output.txt
51            ./s1 > /dev/null
52            echo $7 >> output.txt | echo $7
53            # both to both
54        else
55            ./s1 |& tee output.txt
56            echo $7 >> output.txt | echo $7
57        fi
58    fi
```

Скрипт запускается командой с тремя аргументами:

```
$ ./re_script arg1 arg2 arg3
```

arg1 – путь к программе для запуска

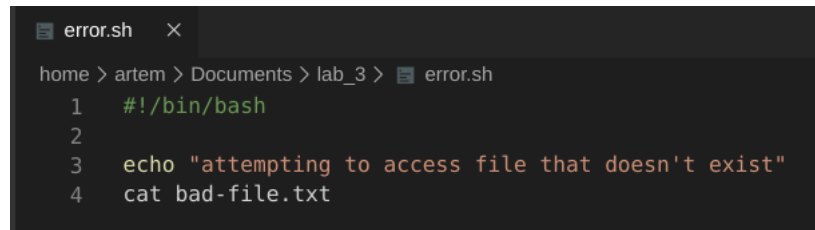
За поток **stdout** отвечает аргумент arg2, за **stderr** – arg3

arg2 и arg3 могут принимать значения:

- **t (terminal)** – поток выводится в терминал
- **f (file)** – поток перенаправляется в файл
- **b (both)** – поток выводится в терминал и дублируется в файл

В конце выводится код завершения. При перенаправлении потоков в файл они выводятся в output.txt

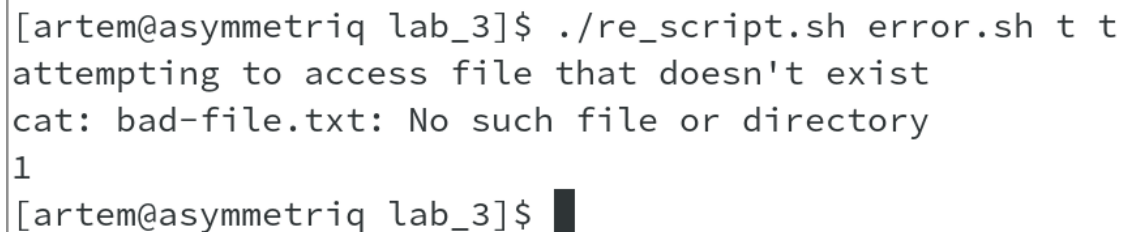
Приведём несколько примеров работы скрипта. Протестируем его на тестовом скрипте error.sh:



```
error.sh x
home > artem > Documents > lab_3 > error.sh
1  #!/bin/bash
2
3  echo "attempting to access file that doesn't exist"
4  cat bad-file.txt
```

Рисунок 1. Содержание тестового скрипта

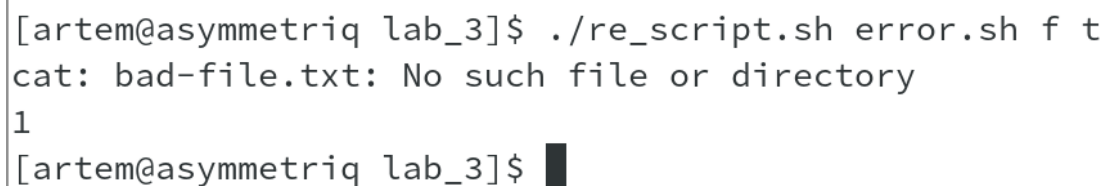
## 1. stdout и stderr в терминал



```
[artem@asymmetriq lab_3]$ ./re_script.sh error.sh t t
attempting to access file that doesn't exist
cat: bad-file.txt: No such file or directory
1
[artem@asymmetriq lab_3]$
```

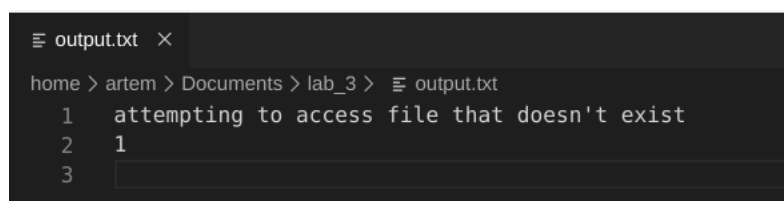
Рисунок 2.  $arg2 = t$ ,  $arg3 = t$  – оба потока выводятся в терминал

## 2. stdout в файл, stderr в терминал



```
[artem@asymmetriq lab_3]$ ./re_script.sh error.sh f t
cat: bad-file.txt: No such file or directory
1
[artem@asymmetriq lab_3]$
```

Рисунок 3.  $arg2 = f$ ,  $arg3 = t$  – stdout выводятся в файл, stderr в терминал



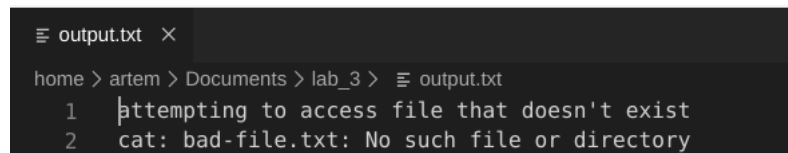
```
output.txt x
home > artem > Documents > lab_3 > output.txt
1  attempting to access file that doesn't exist
2  1
3
```

Рисунок 4. Перенаправление stdout в файл

### 3. stdout и stderr и в терминал, и в файл

```
[artem@asymmetriq lab_3]$ ./re_script.sh error.sh b b
attempting to access file that doesn't exist
cat: bad-file.txt: No such file or directory
```

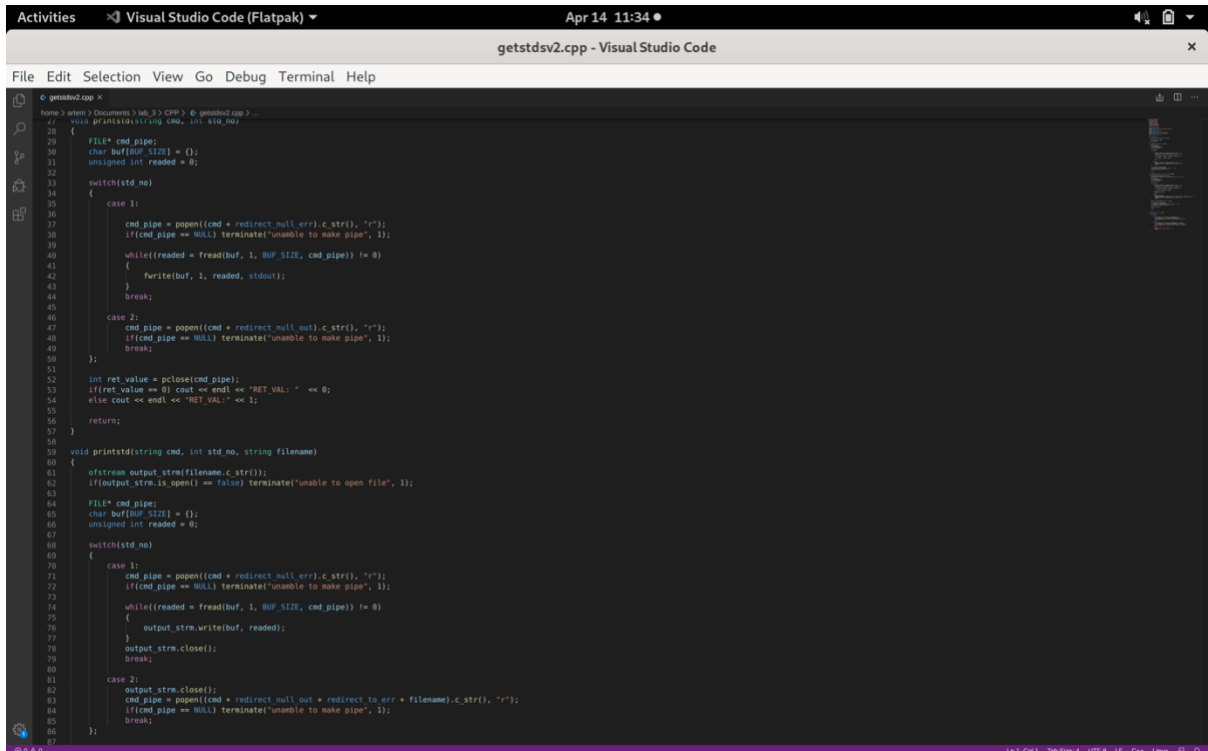
Рисунок 5.  $arg2 = b$ ,  $arg3 = b$  – оба потока выводятся в терминал и дублируются в файл



```
home > artem > Documents > lab_3 > output.txt
1 attempting to access file that doesn't exist
2 cat: bad-file.txt: No such file or directory
```

Рисунок 6. Дублируем оба потока в файл

# ПРИМЕР ИСПОЛЬЗОВАНИЯ ПРИЛОЖЕНИЯ



```
getstdsv2.cpp
// void printstd(string cmd, int std_no)
// {
//     FILE* cmd_pipe;
//     char buf[BUF_SIZE] = {};
//     unsigned int readed = 0;
//     switch(std_no)
//     {
//         case 1:
//             cmd_pipe = popen(cmd + redirect_null_err).c_str(), "r");
//             if(cmd_pipe == NULL) terminate("unable to make pipe", 1);
//             while((readed = fread(buf, 1, BUF_SIZE, cmd_pipe)) != 0)
//             {
//                 fwrite(buf, 1, readed, stdout);
//             }
//             break;
//         case 2:
//             cmd_pipe = popen(cmd + redirect_null_out).c_str(), "r");
//             if(cmd_pipe == NULL) terminate("unable to make pipe", 1);
//             break;
//     }
//     int ret_value = pclose(cmd_pipe);
//     if(ret_value == 0) cout << endl << "RET_VAL: " << 0;
//     else cout << endl << "RET_VAL: " << 1;
//     return;
// }
// void printstd(string cmd, int std_no, string filename)
// {
//     ofstream output_str(filename.c_str());
//     if(output_str.is_open() == false) terminate("unable to open file", 1);
//     FILE* cmd_pipe;
//     char buf[BUF_SIZE] = {};
//     unsigned int readed = 0;
//     switch(std_no)
//     {
//         case 1:
//             cmd_pipe = popen(cmd + redirect_null_err).c_str(), "r");
//             if(cmd_pipe == NULL) terminate("unable to make pipe", 1);
//             while((readed = fread(buf, 1, BUF_SIZE, cmd_pipe)) != 0)
//             {
//                 output_str.write(buf, readed);
//             }
//             output_str.close();
//             break;
//         case 2:
//             output_str.close();
//             cmd_pipe = popen(cmd + redirect_null_out + redirect_to_err + filename).c_str(), "r");
//             if(cmd_pipe == NULL) terminate("unable to make pipe", 1);
//             break;
//     }
// }
```

Приложение запускается с 2-мя или 3-мя аргументами:

```
$ ./getstdsv2 arg1 arg2 (arg3)
```

- `arg1` – путь к программе для запуска
- `arg2` – выбор потока, принимает значения **stdout** и **stderr**
- `(arg3)` – опциональное название выходного файла, если выбранный поток выводится в файл.

В конце выводится код завершения.



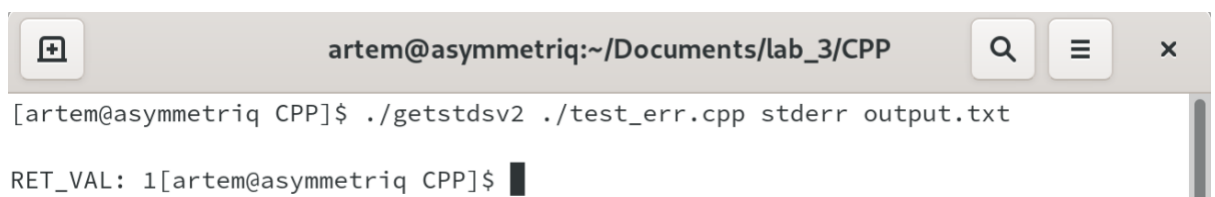
Приведём несколько примеров работы приложения. Протестируем его на приложении test\_err:



```
test_err.cpp ×
home > artem > Documents > lab_3 > CPP > test_err.cpp > ...
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello out";
6      std::cerr << "Hello err";
7
8      return -1;
9  }
10 |
```

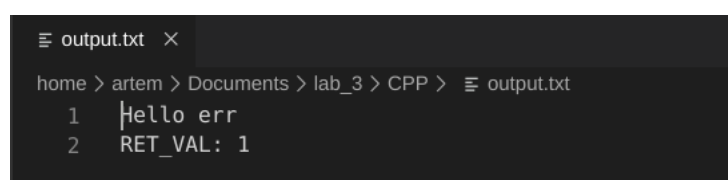
Рисунок 7. Тестовое приложение test\_err

## 1. stderr выводится в файл



```
artem@asymmetriq:~/Documents/lab_3/ CPP
[artem@asymmetriq CPP]$ ./getstdsv2 ./test_err.cpp stderr output.txt
RET_VAL: 1[artem@asymmetriq CPP]$
```

Рисунок 8. Получаем поток stderr и выводим его в файл



```
output.txt ×
home > artem > Documents > lab_3 > CPP > output.txt
1  Hello err
2  RET_VAL: 1
```

Аналогично можем перенаправить в файл поток stdout.

## 2. stdout выводится в терминал

---

```
[artem@asymmetriq CPP]$ ./getstdsv2 ./test_err stdout
Hello out
RET_VAL:1[artem@asymmetriq CPP]$
```

Рисунок 9. Получаем поток stdout и выводим его в терминал

## **ЗАКЛЮЧЕНИЕ**

### **Вывод:**

В ходе выполнения лабораторной работы удалось разработать два варианта приложения для работы с потоками вывода stdout и stderr, а также получить практические навыки.

## ПРИЛОЖЕНИЕ

Исходный код shell-скрипта:

```
#!/bin/bash
# bash script for output redirection

if [ $2 = "f" ]
then
    # out and err to file
    if [ $3 = "f" ]
    then
        ./$1 &> output.txt
        echo $? >> output.txt
    # out to file
    elif [ $3 = "t" ]
    then
        ./$1 > output.txt
        echo $? >> output.txt | echo $?
    # out and err to file + err to terminal
    else
        ./$1 &> output.txt
        ./$1 > /dev/null
        echo $? >> output.txt | echo $?
    fi
elif [ $2 = "t" ]
then
    # out and err to terminal
    if [ $3 = "t" ]
    then
        ./$1
        echo $?
    # err to file
    elif [ $3 = "f" ]
    then
        ./$1 2> output.txt
        echo $? >> output.txt | echo $?
    # out and err to terminal + err to file
    else
        ./$1
        ./$1 2> output.txt > /dev/null
        echo $? >> output.txt | echo $?
    fi
else
    # out to file and terminal + err to file
    if [ $3 = "f" ]
    then
        ./$1 | tee output.txt
        ./$1 > /dev/null 2>> output.txt
        echo $? >> output.txt | echo $?
    # out to file and terminal + err to terminal
    elif [ $3 = "t" ]
    then
        ./$1 | tee output.txt
        ./$1 > /dev/null
        echo $? >> output.txt | echo $?
    # both to both
    else
        ./$1 |& tee output.txt
```

```

        echo $? >> output.txt | echo $?
    fi
fi

```

## Исходный код приложения:

```

#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

#define BUF_SIZE 256
#define ret_val_str "Child proces(cmd) returned "
#define err_str "_err"
#define out_str "_out"
#define redirect_str " 2>&1"
#define redirect_null_err " 2>/dev/null"
#define redirect_null_out " 1>/dev/null"
#define redirect_to_err " 2>"

using namespace std;

void terminate(string err_msg, int ret_val) // выход из программы экстренно
{
    cerr << err_msg << endl;
    exit(ret_val);
}

void printstd(string cmd, int std_no) // печать потока в консоль
{
    FILE* cmd_pipe; // дескриптор файла
    char buf[BUF_SIZE] = {}; // буфер обмена
    unsigned int readed = 0; // кол-во прочитанных байт

    switch(std_no)
    {
        case 1: // если стдаут
            // перенаправление потока
            cmd_pipe = popen((cmd + redirect_null_err).c_str(),
                "r"); // открываем канал(переназначаем поток вывода дочернего
                // процесса(fork()) на поток cmd_pipe
                if(cmd_pipe == NULL) terminate("unable to make
                pipe", 1); // не создался канал - беда

                while((readed = fread(buf, 1, BUF_SIZE, cmd_pipe))
                != 0) // пока читается, читаем
                {
                    fwrite(buf, 1, readed, stdout); // пиши в
                    stdout
                }
                break;

        case 2: // если стэрр
            // открываем поток, перенаправляем вывод в нул, программа
            // выполнится,

```

```

// т.к ее поток ошибок не связан каналом, ошибка выпишется в
теорминал
        cmd_pipe = popen((cmd + redirect_null_out).c_str(),
"r");
        if(cmd_pipe == NULL) terminate("unamable to make
pipe", 1);
        break;
    };

    int ret_value = pclose(cmd_pipe); // возвращаемое значение
    if(ret_value == 0) cout << endl << "RET_VAL: " << 0;
    else cout << endl << "RET_VAL:" << 1;

    return;
}

void printstd(string cmd, int std_no, string filename) // аналогично в файл
{
    ofstream output_strm(filename.c_str());
    if(output_strm.is_open() == false) terminate("unable to open file",
1);

    FILE* cmd_pipe;
    char buf[BUF_SIZE] = {};
    unsigned int readed = 0;

    switch(std_no)
    {
        case 1:
            cmd_pipe = popen((cmd + redirect_null_err).c_str(),
"r"); // тут все по аналогии
            if(cmd_pipe == NULL) terminate("unamable to make
pipe", 1);

            while((readed = fread(buf, 1, BUF_SIZE, cmd_pipe))
!= 0)
            {
                output_strm.write(buf, readed); // запись в
файл
            }
            output_strm.close();
            break;

        case 2: // здесь аналогично, только вывод ошибок
перенаправляем в созданный файл
            output_strm.close();
            cmd_pipe = popen((cmd + redirect_null_out +
redirect_to_err + filename).c_str(), "r");
            if(cmd_pipe == NULL) terminate("unamable to make
pipe", 1);
            break;
    };

    output_strm.open(filename.c_str(), ios::app);

    int ret_value = pclose(cmd_pipe);
    if(ret_value == 0) // пишем возвращаемое значение
    {
        output_strm << endl << "RET_VAL: " << 0;
        cout << endl << "RET_VAL: " << 0;
    }
}

```

```

    else
    {
        output_strm << endl << "RET_VAL: " << 1;
        cout << endl << "RET_VAL: " << 1;
    }

    output_strm.close();

    return;
}

int main(int argc, char* argv[])
{
    switch(argc)
    {
        case 3: // если 3 аргумента
            if(!strcmp("stdout", argv[2])) printstd(argv[1], 1);
            else if(!strcmp("stderr", argv[2]))
                printstd(argv[1], 2);
            else terminate("unknown descriptor, should be stderr
or stdout", 1);
            break;

        case 4: // если 4
            if(!strcmp("stdout", argv[2])) printstd(argv[1], 1,
argv[3]);
            else if(!strcmp("stderr", argv[2]))
                printstd(argv[1], 2, argv[3]);
            else terminate("unknown descriptor, should be stderr
or stdout", 1);
            break;

        default: // если неверное кол-во
            terminate("too few/much arguments", 1);
            break;
    }
}

```