**ОБЩАЯ ТЕОРИЯ**

**1. Жесткие ссылки (Hard Links)**

**Жесткие ссылки** — это механизмы, которые позволяют нескольким именам (файлам) ссылаться на один и тот же инод в файловой системе. **Инод** — это структура данных, которая хранит информацию о файле (например, его данные, права доступа, владельца, и так далее).

- Каждый файл в Linux/Unix-системах имеет свой инод.

- Жесткая ссылка — это дополнительное имя для существующего файла в файловой системе. Это не копия, а просто еще одно имя, которое указывает на тот же инод. Поэтому оба имени (оригинал и жесткая ссылка) фактически указывают на один и тот же физический файл.

**Особенности жестких ссылок:**

- Все жесткие ссылки равноправны, и нет «основного» или «ссылочного» файла.

- Удаление одного из имен (файлов) не удаляет данные, пока не будет удален последний файл, указывающий на инод. Это важно: файлы с жесткими ссылками не исчезнут, пока все ссылки на инод не будут удалены.

- Нельзя создавать жесткие ссылки на директории (за исключением специального случая с точкой и двумя точками в текущей и родительской директориях), так как это создало бы циклические ссылки и проблемы с навигацией по файловой системе.

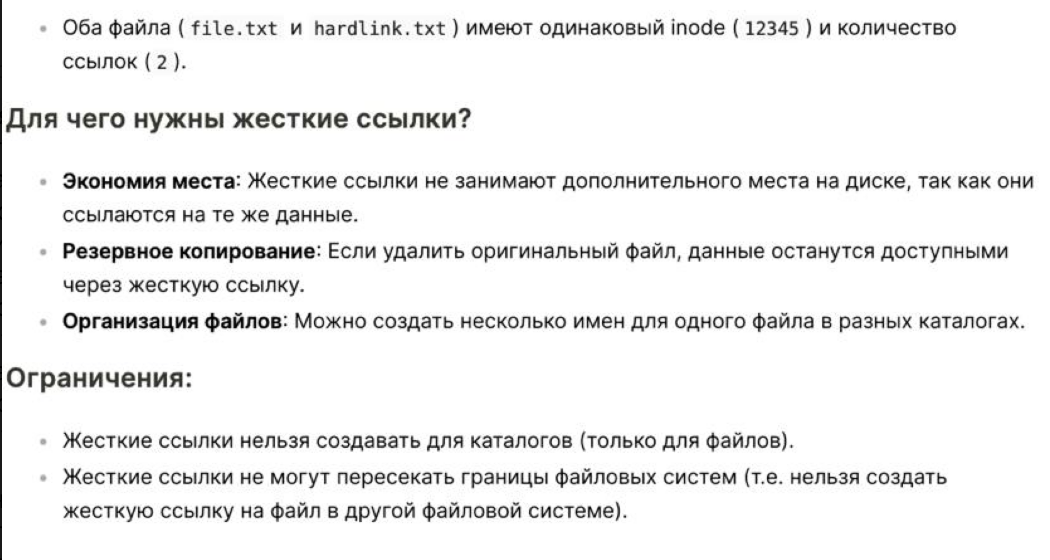
- Жесткие ссылки не могут указывать на файлы в разных файловых системах.

Пример создания жесткой ссылки:

ln original\_file.txt hard\_link.txt

Теперь original\_file.txt и hard\_link.txt — это два разных имени для одного и того же файла.





**2. Символьные ссылки (Symbolic Links или Symlinks)**

**Символьные ссылки** — это специальные файлы, которые содержат путь к другому файлу или директории. Это своего рода ярлыки, которые работают как указатели на другой файл в файловой системе. Символьные ссылки могут указывать на файлы и каталоги в любой части файловой системы, и могут быть использованы для организации удобных и коротких путей.

**Особенности символьных ссылок:**

- Символьная ссылка — это отдельный файл, который просто хранит текстовую строку с путем к другому файлу.

- Символьная ссылка может указывать на файл в другой файловой системе.

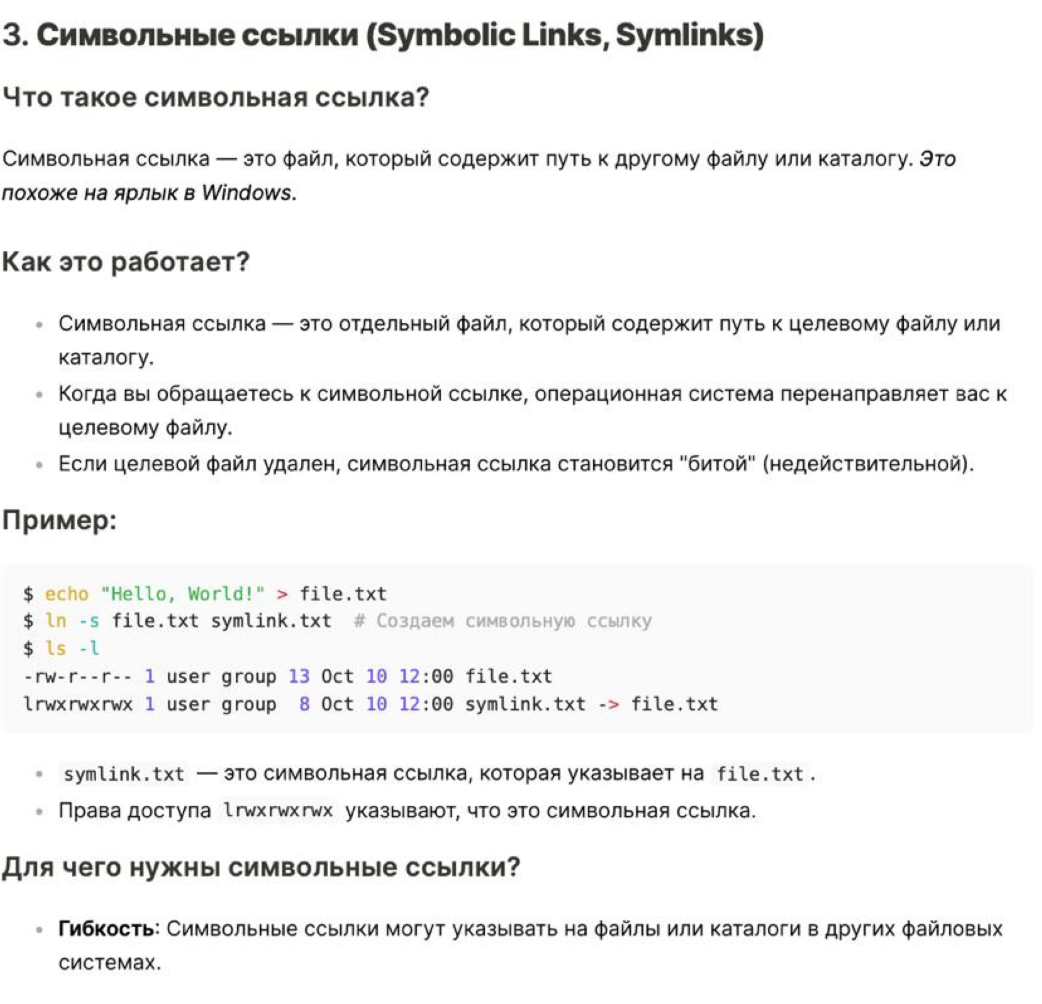
- Если оригинальный файл, на который указывает символьная ссылка, будет удален или перемещен, ссылка перестанет работать, и доступ к файлу будет невозможен. Это называется «битая» ссылка.

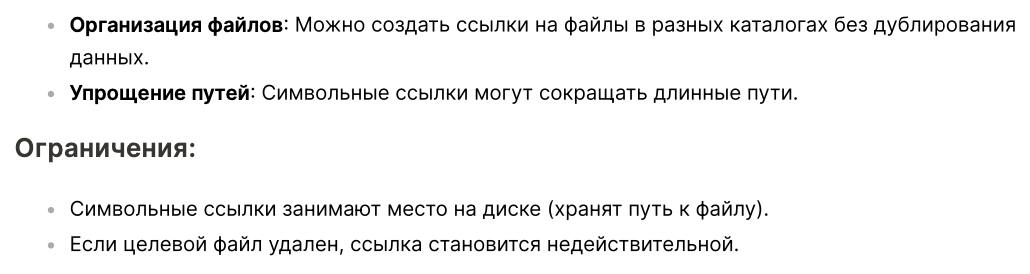
- Символьные ссылки могут быть использованы для создания удобных псевдонимов или для облегчения навигации в сложных структурах каталогов.

Пример создания символьной ссылки:

ln -s original\_file.txt symlink.txt

Теперь symlink.txt будет указывать на original\_file.txt. Если вы откроете symlink.txt, то откроется именно original\_file.txt.



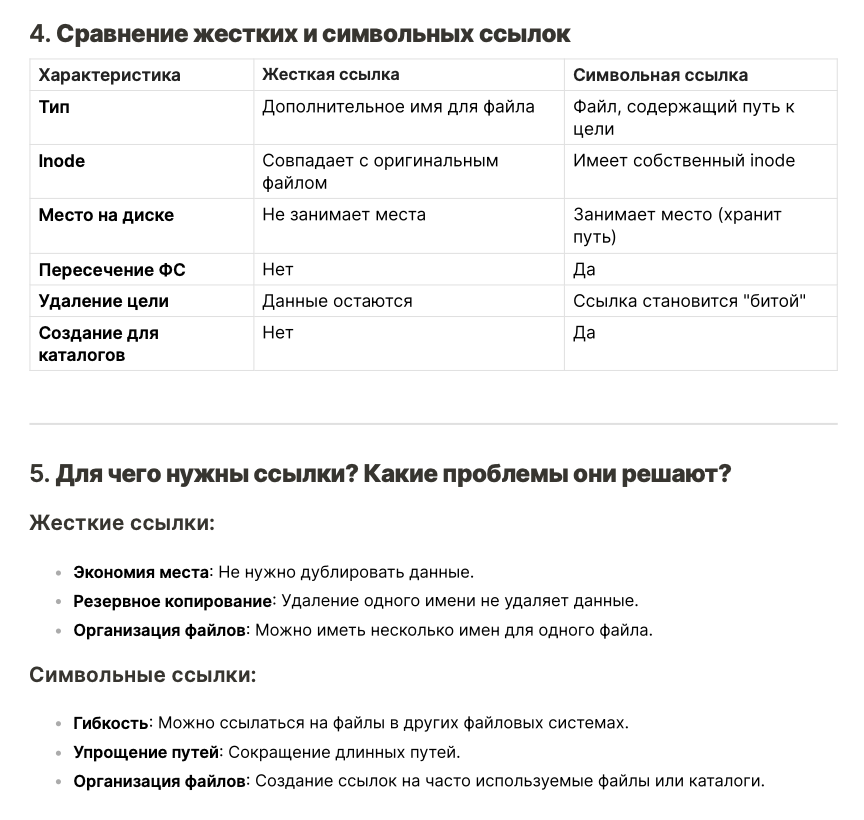


**Основные различия между жесткими и символьными ссылками:**

- **Жесткие ссылки** указывают на один и тот же инод, фактически являясь дополнительными именами для файла. Удаление одного имени не приводит к удалению данных, пока не будет удален последний указатель на инод.

- **Символьные ссылки** — это просто файлы, содержащие путь к другому файлу. Они могут указывать на файлы и каталоги, находящиеся в других местах и на других файловых системах. При удалении файла, на который указывает символьная ссылка, ссылка становится "битой".

- Символьные ссылки можно использовать для создания связей между файлами и каталогами, находящимися в разных местах системы, в отличие от жестких ссылок.



**3. Как удалить файл?**

В Unix/Linux-системах для удаления файлов используется команда rm. Она удаляет ссылки на файл в файловой системе. Если на файл есть другие ссылки (например, жесткие), он не будет удален, пока все ссылки на этот файл не будут удалены.

Пример:

rm file.txt

Этот вызов удаляет файл file.txt, если это была последняя ссылка на него. Если есть другие ссылки (например, жесткие), файл останется в файловой системе, пока не будут удалены все ссылки на него.

**5. Блочные и символические устройства**

**Блочные устройства** — это устройства, с которыми операционная система работает в блоках фиксированного размера. Например, жесткие диски, SSD, CD/DVD-приводы и другие устройства хранения данных. Эти устройства позволяют произвольный доступ к данным, например, для чтения и записи отдельных блоков данных.

Примеры блочных устройств:

- /dev/sda — основной жесткий диск.

- /dev/sdb — второй жесткий диск.

- /dev/sr0 — CD/DVD приводы.

**Символьные устройства** — это устройства, с которыми операционная система работает как с потоком данных, а не с блоками. Доступ к таким устройствам производится посимвольно, без возможности произвольного обращения к данным. Примером символьных устройств являются последовательные порты (например, COM-порты в Windows) или терминальные устройства.

Примеры символьных устройств:

- /dev/tty — терминальные устройства.

- /dev/zero — специальное устройство, которое всегда возвращает нули.

- /dev/random — источник случайных чисел.

**ЗАДАНИЕ**

2. Написать программу, которая создает, читает, изменяет права доступа и удаляет следующие объекты: файлы, каталоги, символьные и жесткие ссылки. Для определения того какая именно функция должна быть исполнена предлагается иметь необходимое количество жестких ссылок на исполняемый файл с именами соответствующими выполняемому действию и в программе выполнять функцию соответствующую имени жесткой ссылки. Программа должна уметь:

a. создать каталог, указанный в аргументе;

b. вывести содержимое каталога, указанного в аргументе;

c. удалить каталог, указанный в аргументе;

d. создать файл, указанный в аргументе;

e. вывести содержимое файла, указанного в аргументе;

f. удалить файл, указанный в аргументе;

g. создать символьную ссылку на файл, указанный в аргументе;

h. вывести содержимое символьной ссылки, указанный в аргументе;

i. вывести содержимое файла, на который указывает символьная ссылка,

указанная в аргументе;

j. удалить символьную ссылку на файл, указанный в аргументе;

k. создать жесткую ссылку на файл, указанный в аргументе;

l. удалить жесткую ссылку на файл, указанный в аргументе;

m. вывести права доступа к файлу, указанному в аргументе и количество

жестких ссылок на него;

n. изменить права доступа к файлу, указанному в аргументе.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

#include <sys/stat.h>

#include <unistd.h>

#include <fcntl.h>

**void** CreateDirectory(**const** **char** \*path) { // Функция для создания каталога

**if** (mkdir(path, 0755) == -1) { // Создаем каталог с правами доступа 0755 (rwxr-xr-x)

perror("Ошибка при создании каталога");

} **else** {

printf("Каталог '%s' успешно создан.\n", path);

}

}

**void** ListDirectory(**const** **char** \*path) { // Функция для вывода содержимого каталога

DIR \*dir = opendir(path); // Открываем каталог

**if** (dir == **NULL**) {

perror("Ошибка при открытии каталога");

**return**;

}

**struct** dirent \*entry;

**while** ((entry = readdir(dir)) != **NULL**) { // Читаем каждый элемент каталога

printf("%s\n", entry->d\_name); // Выводим имя элемента

}

closedir(dir);

}

**void** RemoveDirectory(**const** **char** \*path) { // Функция для удаления каталога

**if** (rmdir(path) == -1) { // Удаляем каталог

perror("Ошибка при удалении каталога");

} **else** {

printf("Каталог '%s' успешно удален.\n", path);

}

}

**void** CreateFile(**const** **char** \*path) { // Функция для создания файла

**int** fd = open(path, O\_CREAT | O\_WRONLY, 0644); // Создаем файл с правами доступа 0644 (rw-r--r--)

**if** (fd == -1) {

perror("Ошибка при создании файла");

} **else** {

close(fd); // Закрываем файл

printf("Файл '%s' успешно создан.\n", path);

}

}

**void** ReadFile(**const** **char** \*path) { // Функция для вывода содержимого файла

**char** buffer[1024];

**int** fd = open(path, O\_RDONLY); // Открываем файл для чтения

**if** (fd == -1) {

perror("Ошибка при открытии файла"); // Выводим сообщение об ошибке, если файл не удалось открыть

**return**;

}

ssize\_t bytes\_read;

**while** ((bytes\_read = read(fd, buffer, **sizeof**(buffer)))) { // Читаем файл по частям

**if** (bytes\_read == -1) {

perror("Ошибка при чтении файла"); // Выводим сообщение об ошибке, если чтение не удалось

**break**;

}

write(STDOUT\_FILENO, buffer, bytes\_read); // Выводим содержимое файла на экран

}

close(fd); // Закрываем файл

}

**void** RemoveFile(**const** **char** \*path) { // Функция для удаления файла

**if** (unlink(path) == -1) { // Удаляем файл

perror("Ошибка при удалении файла"); // Выводим сообщение об ошибке, если что-то пошло не так

} **else** {

printf("Файл '%s' успешно удален.\n", path); // Сообщаем об успешном удалении файла

}

}

**void** CreateSymlink(**const** **char** \*target, **const** **char** \*link\_path) { // Функция для создания символьной ссылки

**if** (symlink(target, link\_path) == -1) { // Создаем символьную ссылку

perror("Ошибка при создании символьной ссылки"); // Выводим сообщение об ошибке, если что-то пошло не так

} **else** {

printf("Символьная ссылка '%s' -> '%s' успешно создана.\n", link\_path, target); // Сообщаем об успешном создании ссылки

}

}

**void** ReadSymlink(**const** **char** \*path) { // Функция для вывода содержимого символьной ссылки

**char** buffer[1024];

ssize\_t len = readlink(path, buffer, **sizeof**(buffer) - 1); // Читаем содержимое символьной ссылки

**if** (len == -1) {

perror("Ошибка при чтении символьной ссылки"); // Выводим сообщение об ошибке, если чтение не удалось

} **else** {

buffer[len] = '\0'; // Добавляем завершающий нулевой символ

printf("Символьная ссылка '%s' указывает на: %s\n", path, buffer); // Выводим содержимое ссылки

}

}

**void** ReadSymlinkTarget(**const** **char** \*path) { // Функция для вывода содержимого файла, на который указывает символьная ссылка

**char** buffer[1024];

ssize\_t len = readlink(path, buffer, **sizeof**(buffer) - 1); // Читаем содержимое символьной ссылки

**if** (len == -1) {

perror("Ошибка при чтении символьной ссылки"); // Выводим сообщение об ошибке, если чтение не удалось

**return**;

}

buffer[len] = '\0'; // Добавляем завершающий нулевой символ

ReadFile(buffer); // Выводим содержимое файла, на который указывает ссылка

}

**void** RemoveSymlink(**const** **char** \*path) { // Функция для удаления символьной ссылки

**if** (unlink(path) == -1) { // Удаляем символьную ссылку

perror("Ошибка при удалении символьной ссылки"); // Выводим сообщение об ошибке, если что-то пошло не так

} **else** {

printf("Символьная ссылка '%s' успешно удалена.\n", path); // Сообщаем об успешном удалении ссылки

}

}

**void** CreateHardlink(**const** **char** \*target, **const** **char** \*link\_path) { // Функция для создания жесткой ссылки

**if** (link(target, link\_path) == -1) { // Создаем жесткую ссылку

perror("Ошибка при создании жесткой ссылки");

} **else** {

printf("Жесткая ссылка '%s' -> '%s' успешно создана.\n", link\_path, target);

}

}

**void** RemoveHardlink(**const** **char** \*path) { // Функция для удаления жесткой ссылки

**if** (unlink(path) == -1) { // Удаляем жесткую ссылку

perror("Ошибка при удалении жесткой ссылки"); // Выводим сообщение об ошибке, если что-то пошло не так

} **else** {

printf("Жесткая ссылка '%s' успешно удалена.\n", path); // Сообщаем об успешном удалении ссылки

}

}

**void** PrintFileInfo(**const** **char** \*path) { // Функция для вывода прав доступа и количества жестких ссылок

**struct** stat st;

**if** (stat(path, &st) == -1) { // Получаем информацию о файле

perror("Ошибка при получении информации о файле");

**return**;

}

printf("Права доступа: %o\n", st.st\_mode & 0777); // Выводим права доступа

printf("Количество жестких ссылок: %lu\n", st.st\_nlink); // Выводим количество жестких ссылок

}

**void** ChangePermissions(**const** **char** \*path, mode\_t mode) { // Функция для изменения прав доступа

**if** (chmod(path, mode) == -1) { // Изменяем права доступа

perror("Ошибка при изменении прав доступа");

} **else** {

printf("Права доступа к файлу '%s' успешно изменены.\n", path);

}

}

**int** main(**int** argc, **char** \*argv[]) {

**if** (argc < 2) {

fprintf(stderr, "Использование: %s <аргументы>\n", argv[0]); // Выводим сообщение об ошибке, если аргументов недостаточно

**return** 1;

}

// Определяем действие на основе имени программы

**const** **char** \*action = strrchr(argv[0], '/'); // Ищем последний символ '/' в имени программы

action = action ? action + 1 : argv[0]; // Получаем имя программы без пути

**if** (strcmp(action, "create\_dir") == 0) {

CreateDirectory(argv[1]); // Создаем каталог

} **else** **if** (strcmp(action, "list\_dir") == 0) {

ListDirectory(argv[1]); // Выводим содержимое каталога

} **else** **if** (strcmp(action, "remove\_dir") == 0) {

RemoveDirectory(argv[1]); // Удаляем каталог

} **else** **if** (strcmp(action, "create\_file") == 0) {

CreateFile(argv[1]); // Создаем файл

} **else** **if** (strcmp(action, "read\_file") == 0) {

ReadFile(argv[1]); // Читаем файл

} **else** **if** (strcmp(action, "remove\_file") == 0) {

RemoveFile(argv[1]); // Удаляем файл

} **else** **if** (strcmp(action, "create\_symlink") == 0) {

CreateSymlink(argv[1], argv[2]); // Создаем символьную ссылку

} **else** **if** (strcmp(action, "read\_symlink") == 0) {

ReadSymlink(argv[1]); // Читаем символьную ссылку

} **else** **if** (strcmp(action, "read\_symlink\_target") == 0) {

ReadSymlinkTarget(argv[1]); // Читаем файл, на который указывает символьная ссылка

} **else** **if** (strcmp(action, "remove\_symlink") == 0) {

RemoveSymlink(argv[1]); // Удаляем символьную ссылку

} **else** **if** (strcmp(action, "create\_hardlink") == 0) {

CreateHardlink(argv[1], argv[2]); // Создаем жесткую ссылку

} **else** **if** (strcmp(action, "remove\_hardlink") == 0) {

RemoveHardlink(argv[1]); // Удаляем жесткую ссылку

} **else** **if** (strcmp(action, "print\_file\_info") == 0) {

PrintFileInfo(argv[1]); // Выводим информацию о файле

} **else** **if** (strcmp(action, "change\_permissions") == 0) {

ChangePermissions(argv[1], strtol(argv[2], **NULL**, 8)); // Изменяем права доступа

} **else** {

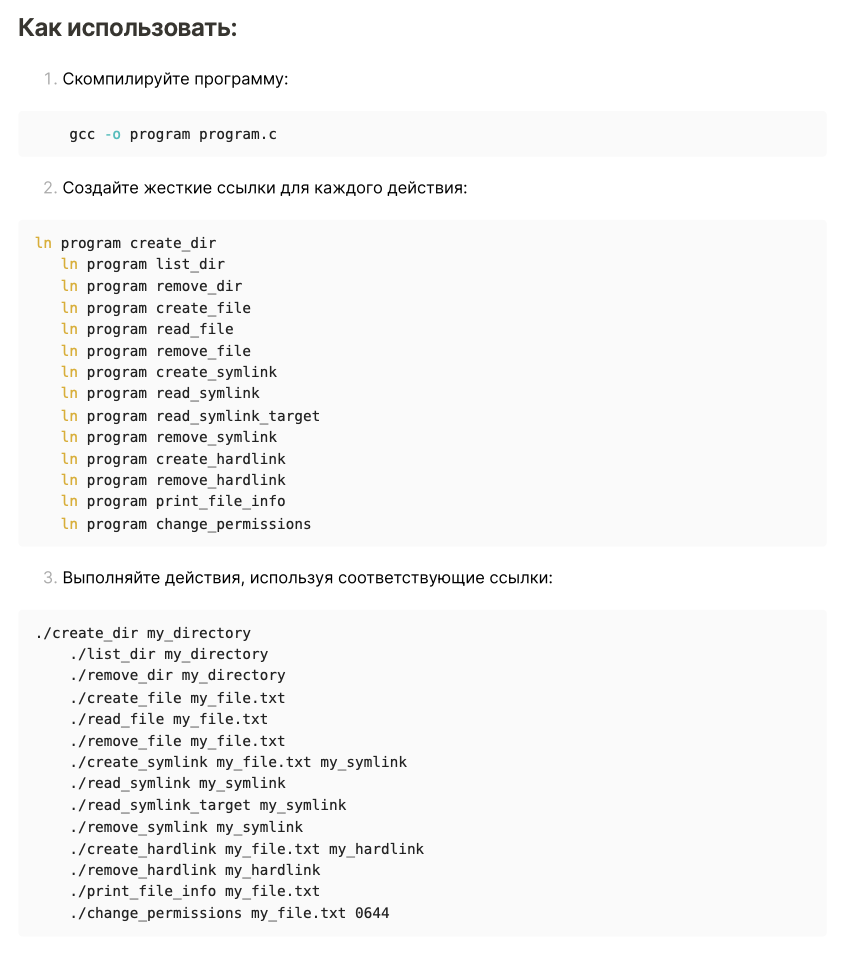
fprintf(stderr, "Неизвестное действие: %s\n", action); // Выводим сообщение об ошибке, если действие неизвестно

**return** 1;

}

**return** 0;

}



**4. Создание символьной ссылки на программу**

Если вы создадите символьную ссылку на вашу программу, например, назовете её create\_dir, она будет работать, если путь к программе правильный. Важно, чтобы символьная ссылка указывала на исполнимый файл (например, бинарник или скрипт). Например:

ln -s /path/to/your/program create\_dir

Если все сделано правильно, и путь в символьной ссылке корректен, то при вызове create\_dir будет запускаться ваша программа.