**Практическая работа № 2.**

1. Регистрация на портале github.com осуществлялась в первой практической работе. Создается новый публичный репозиторий с именем mysyslog.

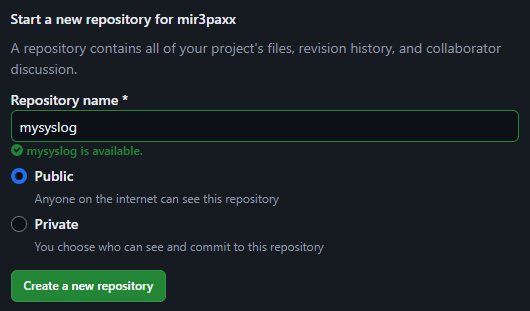


Рис. 1. Создание нового репозитория.

1. Далее требуется настроить файл .gitignore, чтобы из локального репозитория не попадали лишние файлы. Сначала устанавливаются данные, которые помогут идентифицировать автора коммитов.

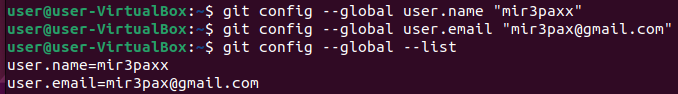


Рис. 2. Идентификация автора коммитов

Затем создается директория proj, в которую будет клонироваться проект. Далее с помощью команды git clone происходит клонирование созданного для практической репозитория на git на локальный компьютер, после выполнения этой команды Git создаст новую директорию с именем mysyslog в текущем рабочем каталоге.

После этого осуществляется командой cd осуществляется переход в директорию клонированного проекта. Затем в этой директории создается файл .gitignore и редактируется с помощью текстового редактора nano.

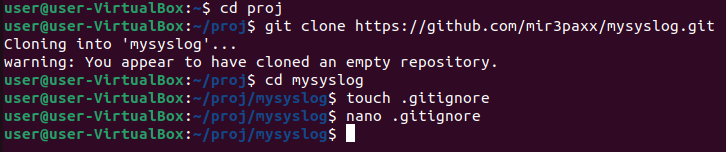


Рис. 3. Клонирование проекта

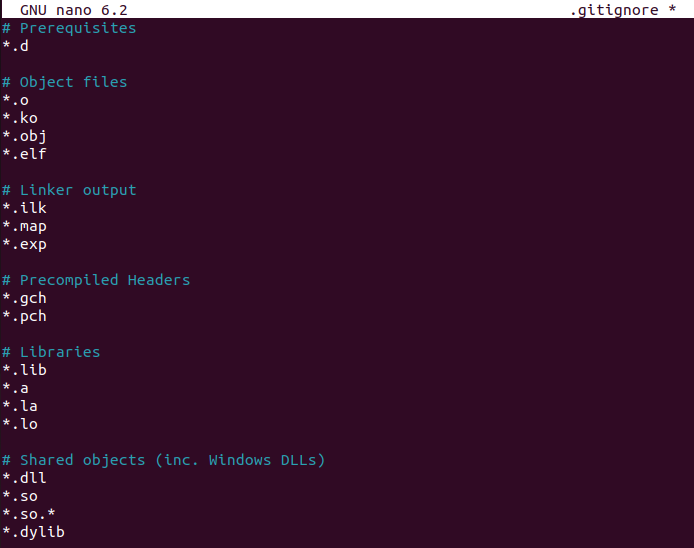


Рис. 4. Содержимое файла .gitignore

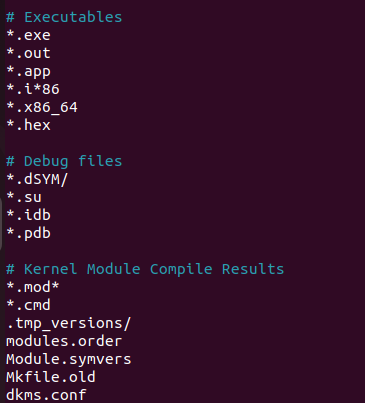


Рис. 5. Содержимое файла .gitignore

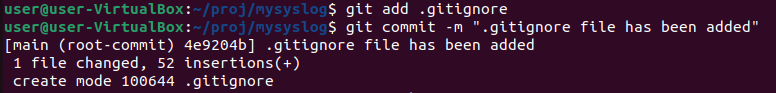


Рис. 6. Добавление и коммит файла .gitignore в репозитории git

Далее, чтобы опубликовать изменения, генерируется персональный токен доступа для аутентификации.

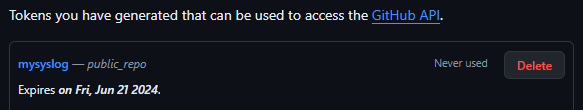


Рис. 7. Создание токена

Затем обновляется URL для удаленного репозитория, чтобы включить созданный токен, команда git push origin main используется для публикации изменений, сделанных в локальном репозитории в ветке main, на удаленный репозиторий Git.

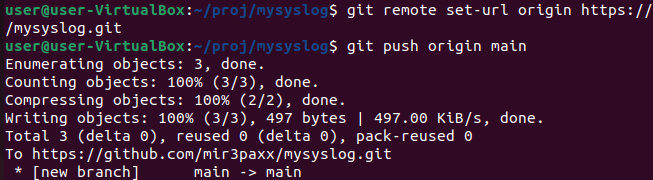


Рис. 8. Публикация изменений

1. Далее нужно добавить в корне проекта его описание в формате Markdown. Для этого создается файл README с расширением md, этот файл написан с использованием синтаксиса Markdown. Заголовки обозначаются символом ‘#’, чем больше символов ‘#’, тем меньше заголовок. С помощью ‘\*\*\*\*’ можно выделить текст жирным. С помощью символа ‘-’ создаются списки.

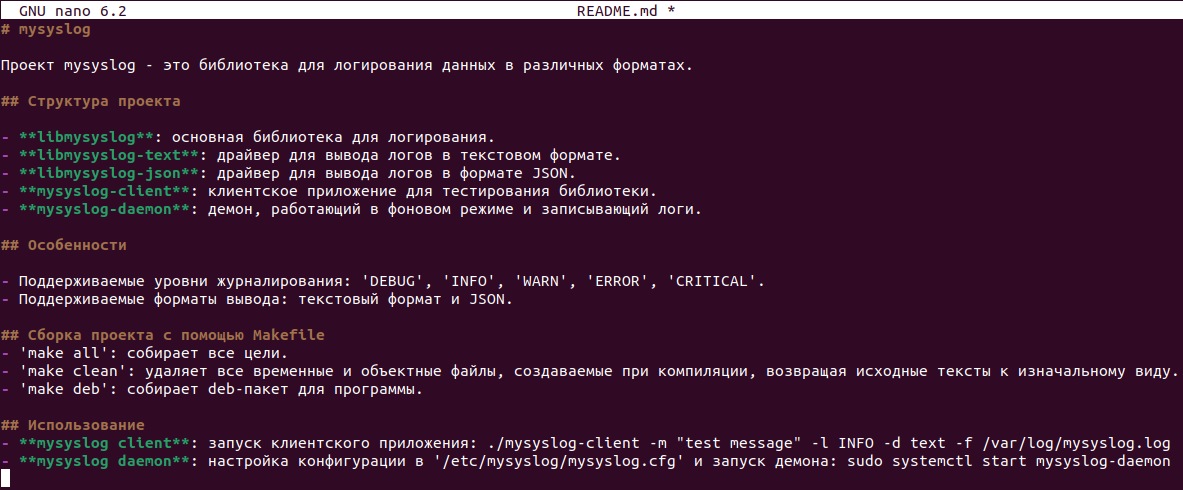


Рис. 9. Создание README файла

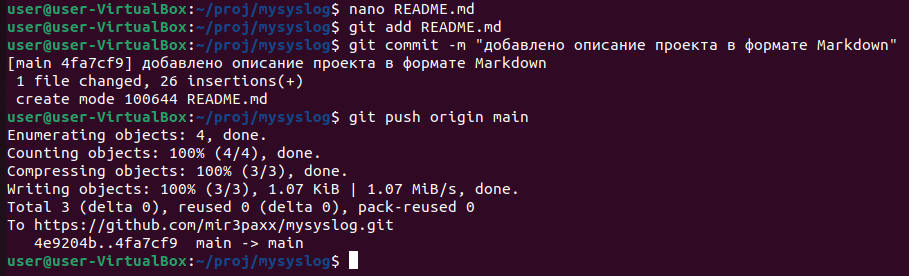


Рис. 10. Добавление README файла в репозиторий на GitHub



Рис. 11. Результат

1. Требуется завести подпроекты: libmysyslog, libmysyslog-text, libmysyslog-json, libmysyslog-client, libmysyslog-daemon.

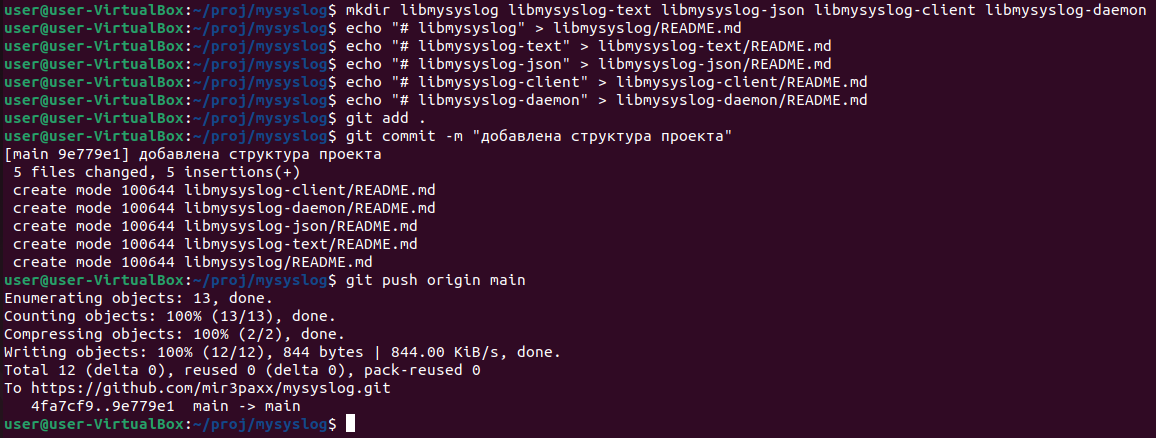


Рис. 12. Создание подпроектов и добавление их в репозиторий на GitHub

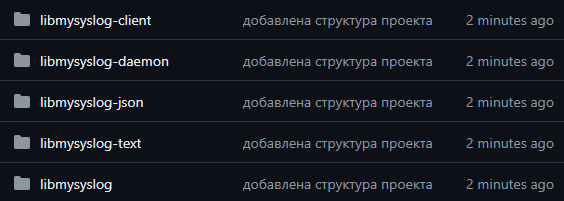


Рис. 13. Успешное создание структуры проекта

1. Далее происходит реализация самого проекта. Сначала устанавливается пакет build-essential, который содержит компилятор gcc.

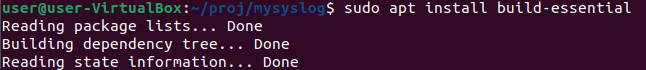


Рис. 14. Установка build-essential

Далее устанавливается пакет для работы с JSON.

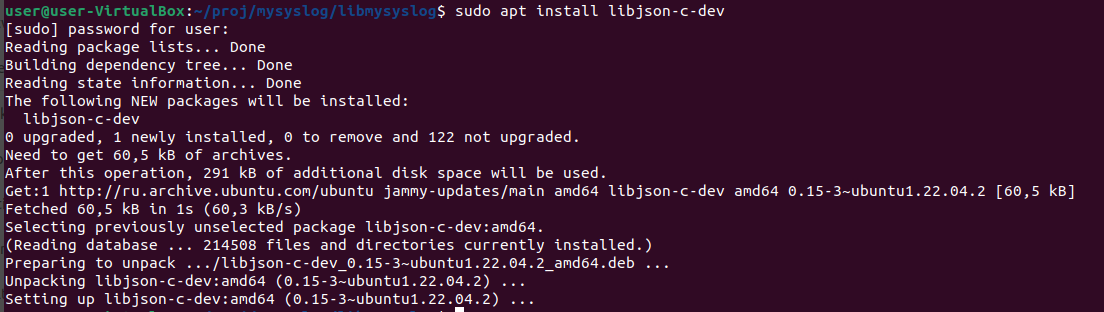


Рис. 15. Установка libjson-c-dev

Затем в директории libmysyslog создается основной файл библиотеки libmysyslog.c. libmysyslog.c реализует функцию журналирования mysyslog, которая записывает сообщения в лог-файл в текстовом или JSON формате.

Сначала функция открывает указанный файл в режиме добавления. Если файл не может быть открыт, функция возвращает -1. Далее функция получает текущее время и преобразует его в строку. Затем преобразует числовое значение уровня логирования в строку и в зависимости от указанного формата (текстовый или JSON), функция записывает сообщение в лог-файл. В конце функция закрывает лог-файл и возвращает 0.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#define DEBUG 0

#define INFO 1

#define WARN 2

#define ERROR 3

#define CRITICAL 4

int mysyslog(const char\* msg, int level, int driver, int format, const char\* path) {

// Реализация функции журналирования

FILE \*log\_file;

log\_file = fopen(path, "a");

if (log\_file == NULL) {

return -1;

}

time\_t now;

time(&now);

char \*timestamp = ctime(&now);

timestamp[strlen(timestamp) - 1] = '\0'; // Удаление символа новой строки

const char \*level\_str;

switch (level) {

case DEBUG: level\_str = "DEBUG"; break;

case INFO: level\_str = "INFO"; break;

case WARN: level\_str = "WARN"; break;

case ERROR: level\_str = "ERROR"; break;

case CRITICAL: level\_str = "CRITICAL"; break;

default: level\_str = "UNKNOWN"; break;

}

if (format == 0) {

// Текстовый формат

fprintf(log\_file, "%s %s %d %s\n", timestamp, level\_str, driver, msg);

} else {

// Формат JSON

fprintf(log\_file, "{\"timestamp\":\"%s\",\"log\_level\":\"%s\",\"driver\":%d,\"message\":\"%s\"}\n", timestamp, level\_str, driver, msg);

}

fclose(log\_file);

return 0;

}

Листинг 1. libmysyslog.c

Затем создается заголовочный файл libmysyslog.h, который содержит объявление функции mysyslog и директивы предохранения от повторного включения (include guards).

#ifndef LIBMYSYSLOG\_H

#define LIBMYSYSLOG\_H

int mysyslog(const char\* msg, int level, int driver, int format, const char\* path);

#endif // LIBMYSYSLOG\_H

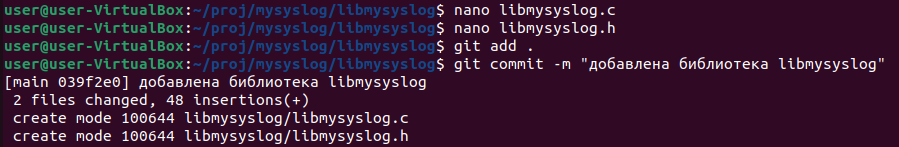
Листинг 2. Заголовочный файл libmysyslog.

Рис. 16. Коммит

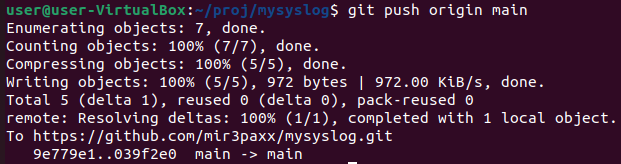


Рис. 17. Публикация изменений

Далее в директории libmysyslog-text создается файл libmysyslog-text.c. Он содержит функцию text\_log, которая оборачивает вызов функции mysyslog с определенными параметрами. Функция text\_log предоставляет упрощенный интерфейс для записи логов в текстовом формате. Она вызывает более универсальную функцию mysyslog с фиксированными значениями для driver и format, что облегчает использование этой функции для задач текстового логирования.

#include "libmysyslog.h"

int text\_log(const char\* msg, int level, const char\* path) {

return mysyslog(msg, level, 0, 0, path);

}

Листинг 3. libmysyslog-text.c

Затем создается заголовочный файл libmysyslog-text.h.

#ifndef LIBMYSYSLOG\_TEXT\_H

#define LIBMYSYSLOG\_TEXT\_H

int text\_log(const char\* msg, int level, const char\* path);

#endif // LIBMYSYSLOG\_TEXT\_H

Листинг 4. libmysyslog-text.h

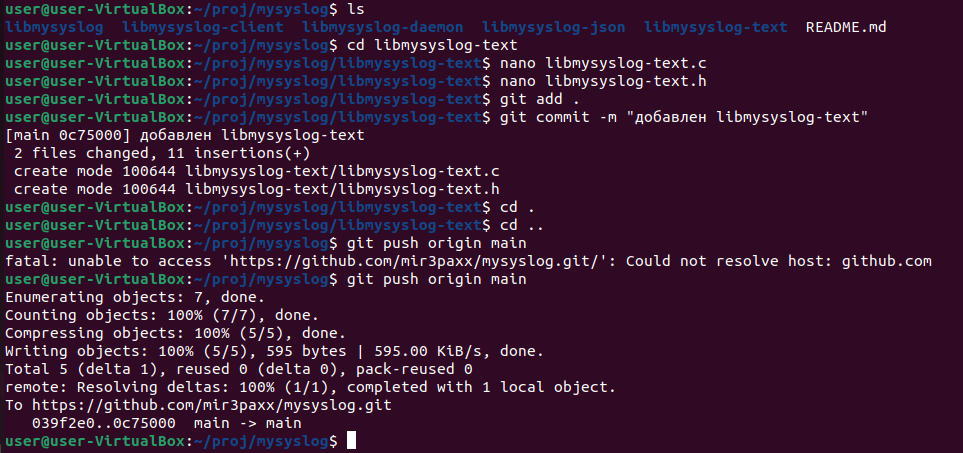


Рис. 18. Публикация изменений

Далее в директории libmysyslog-json создается файл libmysyslog-json.c. Этот код является реализацией функции json\_log, которая используется для записи логов в формате JSON.

#include "libmysyslog.h"

int json\_log(const char\* msg, int level, const char\* path) {

return mysyslog(msg, level, 1, 1, path);

}

Листинг 5. libmysyslog-json.c

Затем создается заголовочный файл libmysyslog-json.h.

#ifndef LIBMYSYSLOG\_JSON\_H

#define LIBMYSYSLOG\_JSON\_H

int json\_log(const char\* msg, int level, const char\* path);

#endif // LIBMYSYSLOG\_JSON\_H

Листинг 6. libmysyslog-json.h

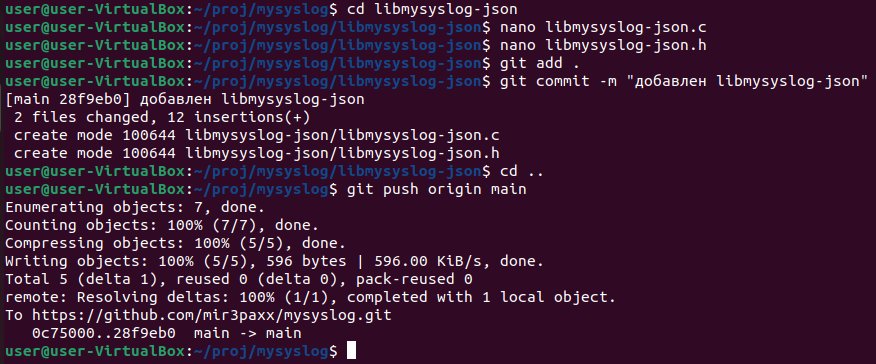


Рис. 19. Публикация изменений

Далее происходит создание приложения libmysyslog-client. В директории libmysyslog-client создается файл libmysyslog-client.c. Эта программа обрабатывает аргументы командной строки, проверяет обязательные параметры и вызывает функцию логирования. Если возникает ошибка, программа выводит соответствующее сообщение и завершает выполнение.

#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#include "libmysyslog.h"

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

int opt;

char \*msg = NULL;

int level = INFO;

int driver = 0;

int format = 0;

char \*path = NULL;

while ((opt = getopt(argc, argv, "m:l:d:f:p:")) != -1) {

switch (opt) {

case 'm': msg = optarg; break;

case 'l': level = atoi(optarg); break;

case 'd': driver = atoi(optarg); break;

case 'f': format = atoi(optarg); break;

case 'p': path = optarg; break;

default: fprintf(stderr, "Usage: %s -m message -l level -d driver -f format -p path\n", argv[0]); exit(EXIT\_FAILURE);

}

}

if (msg == NULL || path == NULL) {

fprintf(stderr, "Message and path are required\n");

exit(EXIT\_FAILURE);

}

if (mysyslog(msg, level, driver, format, path) != 0) {

fprintf(stderr, "Failed to write log\n");

exit(EXIT\_FAILURE);

}

return 0;

}

Листинг 7. libmysyslog-client.c

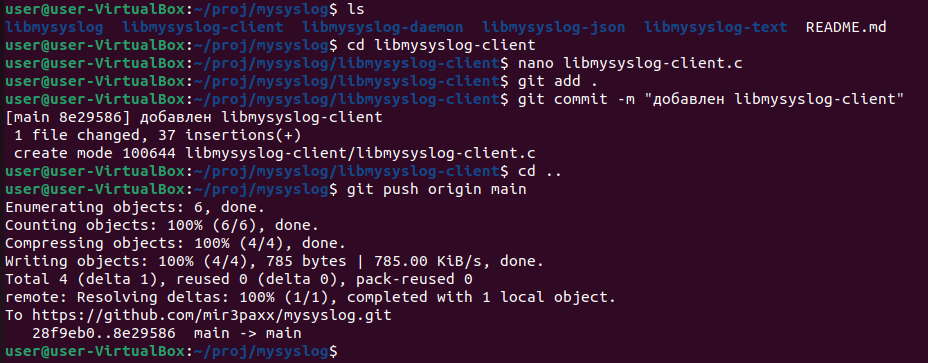


Рис. 20. Публикация изменений

Далее происходит создание приложения демона libmysyslog-daemon. В директории libmysyslog-daemon создается файл libmysyslog-daemon.c. Программа предназначена для автоматического записи сообщений в лог с определенным интервалом времени. Она использует конфигурационный файл для определения пути к файлу лога, формата и драйвера логирования.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include "libmysyslog.h"

#define CONFIG\_FILE "/etc/mysyslog/mysyslog.cfg"

volatile sig\_atomic\_t stop;

void handle\_signal(int signal) {

stop = 1;

}

void read\_config(char\* path, int\* format, int\* driver) {

FILE \*file = fopen(path, "r");

if (file == NULL) {

perror("fopen");

exit(EXIT\_FAILURE);

}

fscanf(file, "path=%s\nformat=%d\ndriver=%d\n", path, format, driver);

fclose(file);

}

int main() {

signal(SIGTERM, handle\_signal);

signal(SIGINT, handle\_signal);

char path[256];

int format;

int driver;

read\_config(CONFIG\_FILE, &format, &driver);

while (!stop) {

mysyslog("Daemon log entry", INFO, driver, format, path);

sleep(5);

}

return 0;

}

Листинг 8. libmysyslog-daemon.c

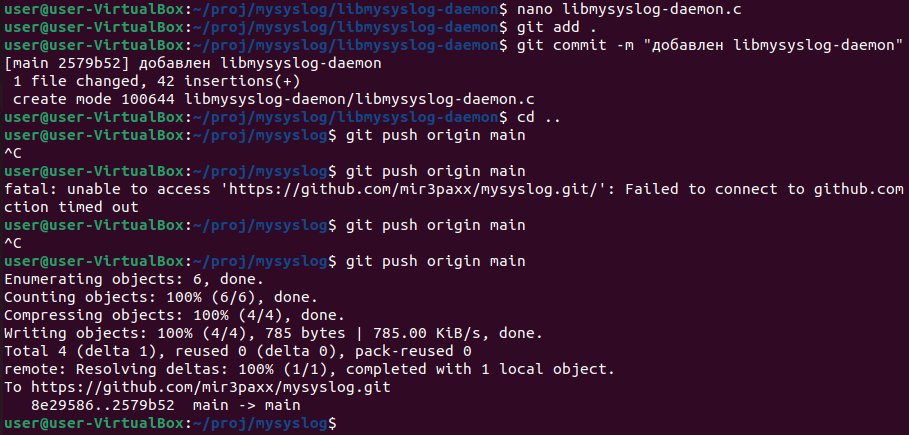


Рис. 21. Публикация изменений

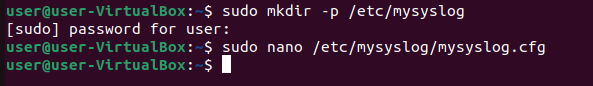


Рис. 22. Создание конфигурационного файла



Рис. 23. Конфигурационный файл

CC = gcc

CFLAGS = -Wall -Wextra -fPIC

TARGET = libmysyslog.so

all: $(TARGET)

$(TARGET): libmysyslog.o

$(CC) -shared -o $(TARGET) libmysyslog.o

libmysyslog.o: libmysyslog.c

$(CC) $(CFLAGS) -c libmysyslog.c

clean:

rm -f $(TARGET) \*.o

.PHONY: all clean

Листинг 9. Makefile для libmysyslog

CC = gcc

CFLAGS = -Wall -Wextra -fPIC

TARGET = libmysyslog-text.so

all: $(TARGET)

$(TARGET): libmysyslog-text.o

$(CC) -shared -o $(TARGET) libmysyslog-text.o

libmysyslog-text.o: libmysyslog-text.c

$(CC) $(CFLAGS) -c libmysyslog-text.c

clean:

rm -f $(TARGET) \*.o

.PHONY: all clean

Листинг 10. Makefile для libmysyslog-text

CC = gcc

CFLAGS = -Wall -Wextra -fPIC

TARGET = libmysyslog-json.so

all: $(TARGET)

$(TARGET): libmysyslog-json.o

$(CC) -shared -o $(TARGET) libmysyslog-json.o

libmysyslog-json.o: libmysyslog-json.c

$(CC) $(CFLAGS) -c libmysyslog-json.c

clean:

rm -f $(TARGET) \*.o

.PHONY: all clean

Листинг 11. Makefile для libmysyslog-json

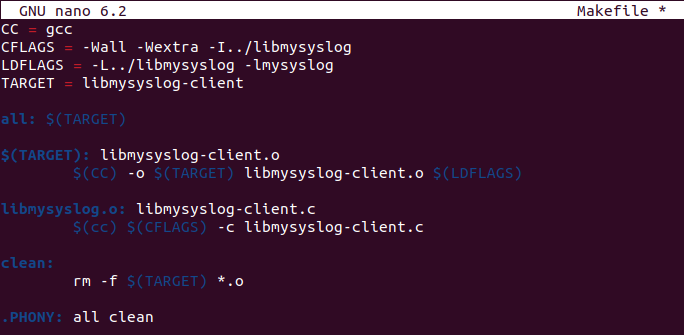


Рис. 24. Makefile для libmysyslog-client

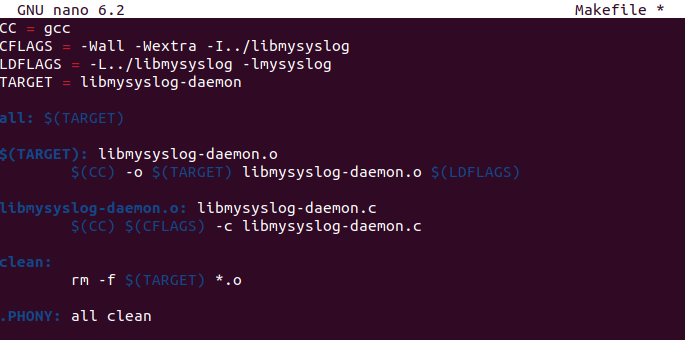


Рис. 25. Makefile для libmysyslog-daemon

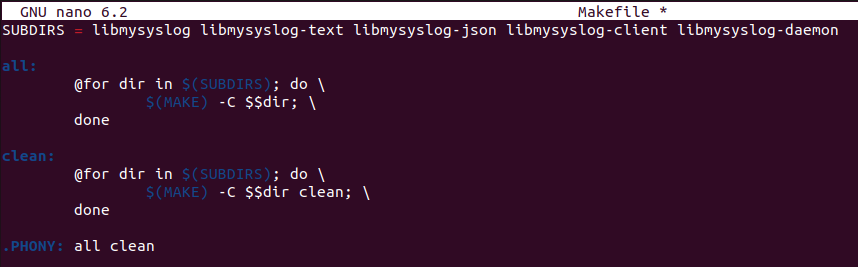


Рис. 26. Общий Makefile

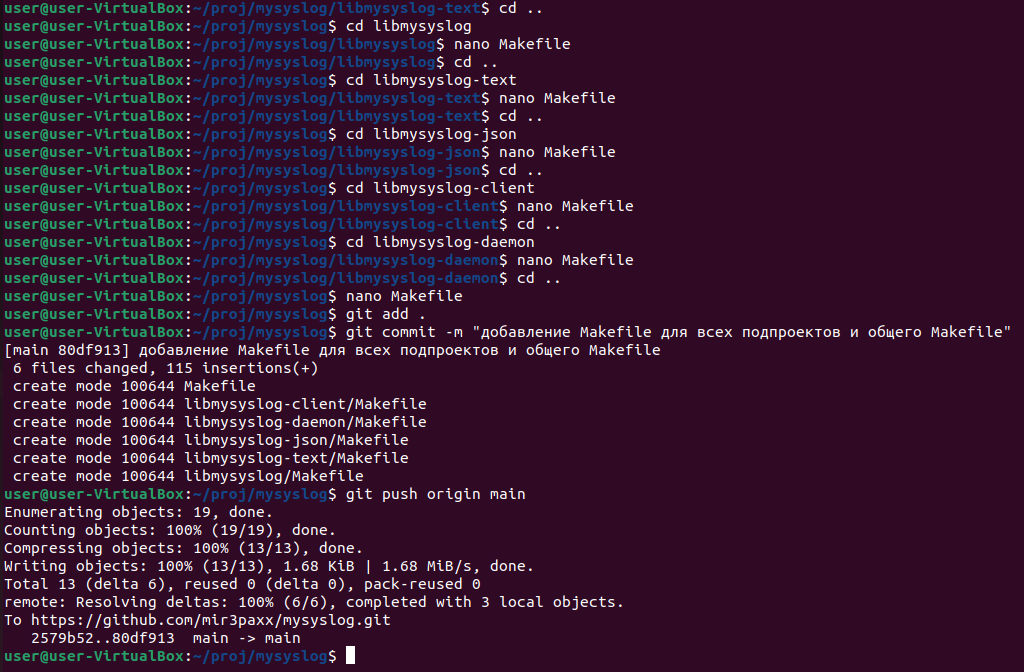


Рис. 27. Публикация изменений

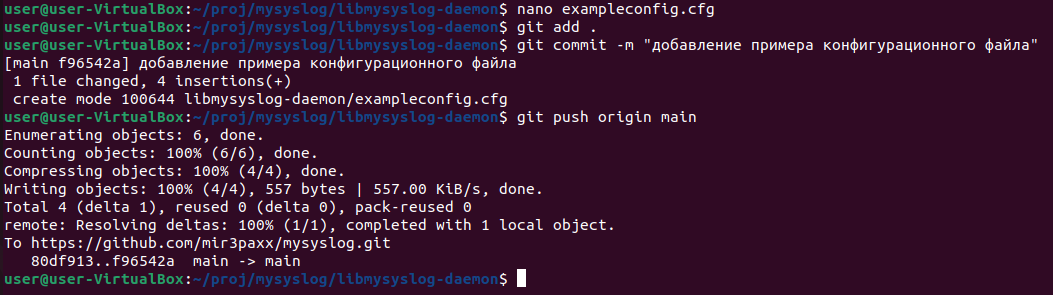


Рис. 27. Создание и публикация примера конфигурационного файла