Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ПРОГРАММА-АНАЛОГ FDUPES С ОГРАНИЧЕНИЕМ ПОИСКА И АНАЛИЗА ПО СИГНАТУРАМ И MIME-ТИПАМ ФАЙЛОВ

БГУИР КР 1-40 02 01 001 ПЗ

Выполнил: студент группы 250541,
Бобрик В. Ю.

Проверил: ст.преподаватель каф. ЭВМ,
Поденок Л. П.

Минск 2025

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____
(подпись)
_____20 г.

З А Д А Н И Е
по курсовому проектированию

Студенту Бобрику Владимиру Юрьевичу

1. Тема проекта Программа-аналог fdupes с ограничением поиска и анализа по сигнатурам и MIME-типам файлов
2. Срок сдачи студентом законченного проекта с 02.06.2025 по 07.06.2025
3. Исходные данные к проекту:
1. Операционная система – Fedora Linux (KDE Plasma);
2. Язык программирования – C (ISO/IEC 9899-2011) с компилятором gcc;
3. Сборка проекта должна выполняться с помощью утилиты make
4. Файлы в составе проекта должны контролироваться git;
5. При отборе дубликатов необходимо определять сигнатуры и MIME-типы файлов, при их несовпадении – исключать возможность дублирования.
4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)
1. Лист задания.
2. Введение.
3. Обзор методов и алгоритмов решения поставленной задачи.
4. Обоснование выбранных методов и алгоритмов.
5. Описание программы для программиста.
6. Описание алгоритмов решения задачи.
7. Руководство пользователя.
8. Заключение.

9. Литература.
10. Приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. Схема алгоритма сравнения файлов по размеру (формат А4).
2. Схема алгоритма сравнения файлов по сигнатурам и MIME-типу (формат А4).
3. Схема алгоритма сравнения файлов по md5-хэшу (формат А4).

6. Консультант по проекту (с обозначением разделов проекта) Л.П. Поденок

7. Дата выдачи задания 26.12.2024

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):
разделы 1, 2 к 31.01.2025 – 20 %;
раздел 3 к 21.02.2025 – 20 %;
раздел 4 к 19.03.2025 – 30 %;
раздел 5 к 16.04.2025 – 10 %;
оформление пояснительной записки к 19.05.2025 - 20 %;
Защита курсового проекта с 12.06 по 02.07

РУКОВОДИТЕЛЬ ст. преподаватель каф. ЭВМ Поденок Л.П
(подпись)

Задание принял к исполнению 26.12.2024
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современную эпоху стремительного роста объемов цифровой информации проблема накопления дублирующихся файлов становится все более острой. На персональных компьютерах, рабочих станциях и серверах часто накапливаются многочисленные копии одних и тех же файлов вследствие резервного копирования, обновления программного обеспечения, несогласованного хранения данных и иных причин. Это ведет к нерациональному использованию дискового пространства, усложняет процессы управления информацией и снижает общую производительность системы.

Целью данного курсового проекта является разработка утилиты, аналогичной существующей `fdupes`, которая способна эффективно выявлять и удалять дублирующиеся файлы в заданном каталоге. Проект предполагает реализацию алгоритмов рекурсивного сканирования файловой системы, фильтрацию данных по параметрам, вывод списка дубликатов, а также обеспечение возможности интерактивного удаления найденных дубликатов.

Задачи проекта включают:

- разработку алгоритма сканирования каталога с учетом рекурсии;
- реализацию эффективного способа группировки файлов по критериям дублирования;
- интеграцию дополнительных фильтров для исключения из рассмотрения файлов, не удовлетворяющих заданным ограничениям;
- создание гибкого интерфейса командной строки с поддержкой различных флагов, позволяющего пользователю настроить режим работы утилиты;
- обеспечение надежной и безопасной процедуры удаления дубликатов с возможностью интерактивного подтверждения или автоматического выполнения операций.

1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Анализ проблемы дублирования файлов

Повторяющиеся копии одного и того же файла могут возникать по множеству причин: от случайных копирований и резервного копирования до ошибок при обновлении и синхронизации данных. Это приводит к ряду негативных последствий, которые существенно влияют на эффективность работы системы.

Во-первых, нерациональное использование дискового пространства является основной проблемой дублирования файлов. Хранение идентичных данных в разных местах системы не только снижает свободное место на носителе, но и увеличивает время резервного копирования, понижает производительность и может затруднять оптимальное распределение ресурсов.

Во-вторых, наличие множества копий одного и того же файла усложняет задачи администрирования и поиска нужной информации. Системным администраторам и конечным пользователям приходится тратить дополнительное время на ручной анализ и очистку файловой системы от ненужных копий, что приводит к дополнительным трудозатратам и снижению эффективности работы.

При анализе большого числа файлов необходимо быстро и без избыточного расхода ресурсов определять потенциальные дубликаты. Наиболее простая стратегия – предварительная фильтрация по размеру файла – позволяет значительно сократить число последующих сравнений. Однако даже после группировки по размеру остается задача точного определения идентичности содержимого файлов. Для этого популярны методы вычисления хэш-сумм (например, `MD5` или `SHA`), позволяющие сравнивать файлы по

характерному «отпечатку». Эти методы имеют свои ограничения: возможны коллизии, а вычисление хэш-суммы для больших файлов требует значительных вычислительных ресурсов. Иногда для окончательной проверки может понадобиться побайтовое сравнение, которое, хоть и является самым точным, также затратное по времени.

Кроме того, при разработке программного обеспечения для поиска дубликатов важно учитывать особенности организации файловой системы. Необходимо корректно обрабатывать рекурсивное сканирование директорий, учитывать символьные и жесткие ссылки, а также обходить потенциально недоступные или защищенные участки файловой системы. Неправильное обращение с этими аспектами может привести к ошибкам в работе программы или даже к повреждению данных.

Таким образом, проблема дублирования файлов имеет как практическое, так и теоретическое значение. С одной стороны, устранение избыточности данных помогает оптимизировать использование дискового пространства и облегчить управление информацией. С другой стороны, сама задача требует аккуратного балансирования между скоростью работы алгоритма, точностью идентификации дубликатов и минимизацией нагрузки на систему.

1.2 Обзор существующих решений

Существует несколько инструментов для поиска и удаления дублирующихся файлов, отличающихся по функциональности, интерфейсу и оптимизации. Рассмотрим подробнее три популярных решения.

1. `fdupes` – это одна из наиболее известных утилит для поиска дубликатов в `Unix`-подобных системах. Основные возможности `fdupes` включают:

– рекурсивное сканирование каталогов, что позволяет находить дубликаты в любых уровнях вложенности;

– группировку файлов по размеру в качестве предварительного этапа, что значительно снижает количество последующих сравнений;

– вычисление хэш-сумм для подтверждения идентичности файлов, что уменьшает вероятность ложных срабатываний;

– поддержку интерактивного режима удаления, где пользователь может выбирать, какой файл сохранить, а какие удалить, и неинтерактивного режима, автоматически оставляющего первый файл из группы и удаляющего остальные.

Недостатками `fdupes` являются ограниченные возможности по настройке (например, отсутствие сложных фильтров) и отсутствие графического интерфейса, что может затруднить использование для пользователей, предпочитающих визуальные редакторы.

2. `dupeGuru` – предлагает кроссплатформенное графическое приложение для поиска дубликатов. Его ключевые особенности:

– интуитивно понятный графический интерфейс, позволяющий пользователям легко запускать сканирование и просматривать результаты без необходимости работы через командную строку;

– возможность сортировать и фильтровать результаты, что облегчает процесс принятия решения о дальнейших действиях с найденными дубликатами;

– расширенные алгоритмы, включающие возможности поиска по схожести изображений, что делает утилиту полезной для работы с медиафайлами.

Однако для `dupeGuru` характерна более высокая потребность в системных ресурсах, и его установка может требовать дополнительных зависимостей, что делает его менее удобным для серверных или минималистичных систем.

3. rmlint — это современное решение, ориентированное на высокую производительность и масштабируемость:

- обладает высокой скоростью работы благодаря оптимизированным алгоритмам и эффективной обработке большого количества файлов;
- предлагает расширенные возможности фильтрации: помимо выявления дубликатов, можно обнаруживать мусор (пустые файлы, битые ссылки и т.д.) и проводить комплексную очистку системы;
- реализует как интерактивный, так и неинтерактивный режимы удаления, что делает его гибким для автоматизации.

Недостаток rmlint заключается в том, что продвинутый функционал может усложнить использование для начинающих пользователей, поскольку командная строка утилиты имеет множество опций, требующих детального изучения.

Каждое из данных решений имеет свои преимущества и недостатки. fdupes отличается простотой и надежностью, что делает его популярным выбором для быстрого поиска дублированных файлов в Unix-системах. dupeGuru нацелен на пользователей, которым важна визуальная интуитивность и расширенные возможности работы с медиафайлами, а rmlint предоставляет мощный инструментарий для комплексной очистки системы в автоматизированном режиме.

1.3 Методы сравнения и идентификации дубликатов

При решении задачи поиска дублирующихся файлов используются многослойные подходы, позволяющие последовательно уменьшать множество кандидатов и повышать точность идентификации. Основные этапы включают:

1) Предварительная фильтрация по размеру файла. На этом этапе файлы группируются по их размеру, поскольку файлы с различными размерами не

могут быть идентичными. Этот метод позволяет быстро отсеять значительное число файлов и сократить количество последующих вычислений.

2) Сравнение MIME-сигнатур. MIME-сигнатуры представляют собой специализированные «отпечатки» содержимого, определяемые по первым байтам файла. Сравнение MIME-сигнатур позволяет не только удостовериться в идентичности файлов, но и проверить, что файлы принадлежат к одному и тому же типу данных. Это особенно полезно, если файлы имеют одинаковые размеры и, возможно, даже совпадающие хэш-суммы, но могут различаться по внутренней структуре или формату. Сравнение MIME-сигнатур помогает обнаружить случаи, когда под внешним сходством файлов (например, за счет похожих размеров) скрываются существенные различия, что повышает надежность алгоритма идентификации.

3) Вычисление хэш-сумм. Для файлов, имеющих одинаковый размер, рассчитываются хэш-суммы с использованием таких алгоритмов, как MD5, SHA-1 или SHA-256. Совпадение хэш-сумм является сильным индикатором идентичности содержимого, хотя вероятность коллизий не может быть полностью исключена, особенно при использовании менее современных алгоритмов.

4) Побайтовое сравнение. В случаях, когда требуется абсолютно точная проверка, выполняется побайтовое сравнение содержимого файлов. Этот метод является самым надежным, но также и самым ресурсоемким, поэтому применяется только на финальном этапе проверки уже предварительно отфильтрованных групп файлов.

Таким образом, комплексный подход позволяет создать надежную систему для выявления дублирующихся файлов, минимизируя вероятность ошибок при идентификации и обеспечивая высокую точность анализа данных.

2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ

В результате анализа различных подходов к поиску и идентификации дублирующихся файлов, с учетом требований по производительности, точности и удобству использования, в данном проекте выбран следующий многоступенчатый алгоритм:

1. Предварительная фильтрация по размеру файлов.

На этом начальном этапе файлы группируются по размеру. Поскольку файлы с различным объемом данных гарантированно не совпадают, данный метод позволяет быстро отсеять большинство кандидатов для дальнейшей детальной проверки. Этот этап требует минимальных вычислительных ресурсов и значительно сокращает число последующих сравнений.

2. Сравнение MIME-сигнатур.

Перед вычислением хэш-сумм проводится сравнение MIME-сигнатур, что позволяет предварительно удостовериться, что файлы принадлежат к одному и тому же типу содержимого. Определение MIME-типа по первым байтам файла позволяет отсеять файлы, имеющие различное форматы, и, таким образом, избежать лишних затрат вычислительных ресурсов на последующие этапы.

3. Вычисление хэш-сумм.

Для файлов, прошедших предыдущие этапы, производится вычисление хэш-сумм с использованием алгоритма MD5. Совпадение хэш-сумм является сильным индикатором идентичности содержимого, что позволяет сформировать группы потенциальных дубликатов с высокой точностью.

4. Побайтовое сравнение.

В случаях, когда требуется абсолютно точное подтверждение идентичности, выполняется побайтовое сравнение содержимого файлов, уже сгруппированных по совпадению хэш-сумм. Хотя этот метод является самым

надежным, он также и наиболее ресурсоемкий, поэтому применяется только на финальном этапе проверки.

Для дальнейшей обработки дублирующихся файлов предусматриваются два режима удаления:

– интерактивный режим:пользователь вручную выбирает, какой файл сохранить из каждой группы дубликатов, что позволяет избежать случайного удаления нужных данных.

– неинтерактивный режим: утилита автоматически сохраняет первый файл каждого набора и удаляет все последующие, что существенно ускоряет процесс при массовом удалении.

Такой многоступенчатый подход, основанный на предварительной фильтрации, сравнении MIME-сигнатур, вычислении хэш-сумм и, при необходимости, побайтовом сравнении, обеспечил высокую эффективность и точность работы утилиты при обработке больших объемов данных. Реализация как интерактивного, так и неинтерактивного режимов удаления дубликатов дополнительно повышает гибкость применения программы, позволяя адаптировать ее к различным сценариям использования — от ручного контроля до автоматизированных процессов очистки. В совокупности принятые решения позволяют обеспечить непревзойденное соотношение между производительностью, точностью и удобством использования, что является ключевым требованием для современных средств оптимизации хранения информации.

Каждый из этих этапов разработан с учетом принципов эффективности и масштабируемости. Рекурсивное сканирование файловой системы, используемое для формирования полного списка файлов, имеет линейную сложность $O(n)$, где n – общее число файлов. На этом же этапе предварительная фильтрация по размеру позволяет быстро отсеять файлы,

гарантирующе отличающиеся по объему, что существенно снижает количество объектов для последующего, более трудоемкого анализа.

Далее, сравнением MIME-сигнатур производится проверка типа данных для каждой записи. Обработка MIME-сигнатуры требует считывания лишь первых нескольких байтов файла, что обеспечивает практически постоянное время работы O(1) для каждого файла. Это позволяет заранее отсеять файлы, относящиеся к различным форматам, не подвергая их дальнейшему анализу и тем самым экономя затраты вычислительных ресурсов.

Следующим этапом является вычисление хэш-сумм для файлов, прошедших предыдущие фильтры. Здесь основным алгоритмом являются современные методы хеширования, в данной работе MD5. Несмотря на то, что вычисление хэша требует обработки всего содержимого файла (то есть имеет сложность O(m) при размере m файла), число файлов, подлежащих этому этапу, значительно снижается благодаря ранней фильтрации. Для окончательной проверки и устранения возможных коллизий применяется побайтовое сравнение уже сформированных групп файлов. Этот этап, отчасти самый ресурсоемкий, вызывается лишь для минимального числа кандидатов, что гарантирует абсолютную точность идентификации дубликатов без значительного увеличения общего времени работы.

Показатели эффективности демонстрируют линейное растущее время работы в зависимости от числа файлов при условии эффективной предварительной фильтрации и группировки. Использование динамических структур данных для хранения информации о файлах позволяет оптимизировать расход оперативной памяти, а многократное сокращение числа кандидатов на каждом этапе обеспечивает высокую производительность даже при масштабном сканировании файловых систем.

Выбранный алгоритм, обеспечивает оптимальное соотношение между точностью идентификации дубликатов и быстродействием.

Разработанный пользовательский интерфейс утилиты сконцентрирован на командной строке, что является наиболее классическим и привычным решением для работы в Unix-подобных системах. Такой подход позволяет обеспечить гибкость и скорость взаимодействия, а также интегрировать приложение в существующие сценарии автоматизации и скриптования. Набор поддерживаемых опций и флагов, таких как режимы интерактивного и неинтерактивного удаления, а также дополнительные параметры фильтрации, обеспечивает точную настройку работы утилиты под конкретные задачи пользователя. Это позволит как системным администраторам, так и пользователям с разным уровнем подготовки быстро адаптировать утилиту под свои требования.

Кроме того, реализация подробного режима вывода (verbose) способствует прозрачности процесса и позволяет пользователю отслеживать все этапы работы программы, что особенно важно при выполнении операций, связанных с удалением файлов. Интерактивный режим удаления, предполагающий явное подтверждение каждого шага, предоставляет дополнительную защиту от случайного удаления важных данных, повышая надежность системы и доверие пользователя к утилите. В свою очередь, неинтерактивный режим предназначен для сценариев массовой очистки, где вмешательство человека минимально, что значительно ускоряет процесс при обработке большого объема информации.

Выбранный интерфейс сочетает в себе простоту, мощность и возможность тонкой настройки, отвечая требованиям как опытных специалистов, так и пользователей, не имеющих глубоких технических знаний. Такое решение облегчает интеграцию утилиты в существующие процессы управления данными, что делает ее универсальным и эффективным инструментом для оптимизации хранения и обработки информации.

3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

3.1 Структура проекта

Проект состоит из набора файлов и директорий, каждый из которых играет определённую роль в реализации утилиты. Ниже приведено описание основных компонентов.

`main.c` – файл, служащий точкой входа в программу. Содержит функцию `main()`. Здесь вызывается `parse_arguments()` для разбора аргументов командной строки, после чего полученные настройки присваиваются глобальным переменным, используемым другими модулями. Затем происходит инициализация процесса сканирования каталогов для поиска файлов посредством вызовов функций соответствующего модуля, запускается основная логика обработки найденных файлов (включающая этапы фильтрации, сравнения и удаления дубликатов), а по завершении работы выполняется очистка выделенной памяти и корректное завершение программы с возвратом соответствующего кода выхода.

`args.h` – заголовочный файл, который является ядром модуля обработки аргументов командной строки. В нем определяется структура `Options`, предназначенная для хранения настроек, переданных пользователем при запуске утилиты, а также объявляются прототипы функций, отвечающих за разбор аргументов. Подключение данного файла в другие части программы обеспечивает единообразную обработку входных параметров и позволяет централизованно управлять настройками, такими как рекурсивный режим, подробный вывод и ограничения по размеру файлов.

`args.c` – реализация функции `parse_arguments()`, которая непосредственно анализирует аргументы командной строки. Функция обрабатывает входящие параметры, извлекает значения и заполняет структуру `Options`, определенную в `args.h`. Дополнительно выполняется проверка

корректности введенных данных, обработка ошибок и установка значений по умолчанию для параметров, не заданных пользователем. Разбор аргументов гарантирует, что дальнейшая работа утилиты основана на валидных и правильно интерпретированных настройках.

`selection.c` – в файле реализована логика обхода заданного каталога и сбор информации о файлах для дальнейшей обработки на предмет дублирования. Здесь происходит рекурсивное сканирование каталогов (при необходимости, если задан соответствующий флаг) с использованием системных вызовов для открытия директорий, чтения их содержимого и получения метаданных файлов (через `stat` и подобные функции). При этом каждый найденный файл обрабатывается на раннем этапе и его характеристики сохраняются в динамически формируемой структуре для дальнейшей работы.

Особое внимание в `selection.c` уделено обработке флагов `-G` и `-L`, которые отвечают за фильтрацию файлов по их размеру. Флаг `-G` задает минимальный размер файла для включения в выборку, а `-L` определяет максимальный допустимый размер. В процессе обхода каталогов для каждого найденного файла производится сравнение его размера с установленными лимитами: если размер файла меньше минимального значения или превышает максимальное, он исключается из дальнейшей обработки.

`selection.c` также занимается группировкой файлов, что обеспечивает эффективную передачу отфильтрованного набора данных на последующие этапы проверки – сравнение `MIME`-сигнатур, вычисление хэш-сумм и, при необходимости, побайтовое сравнение. Таким образом, правильно реализованная логика обхода и первичной фильтрации в `selection.c` не только ускоряет работу утилиты, но и существенно оптимизирует расход вычислительных ресурсов путем раннего устранения файлов, не подходящих для дальнейшего анализа.

filter_size.c – модуль фильтрации файлов по размеру. Функции в filter_size.c исключают из обработки те файлы, которые имеют уникальный размер. Такой подход существенно сокращает объем данных, подлежащих дальнейшему анализу, и оптимизирует общий процесс поиска дубликатов.

filter_mime.c – модуль анализа и сравнения MIME-сигнатур файлов. На этом этапе производится считывание первых байтов каждого файла для определения его типа содержимого, что позволяет предварительно исключить файлы с различной структурой до выполнения более ресурсоемких операций, таких как вычисление хэш-сумм. Такой предварительный отбор по MIME-сигнатурам помогает оптимизировать последующие этапы обработки, гарантируя, что сравниваются только файлы одного и того же типа.

filter_hash.c – модуль вычисления хэш-сумм файлов с использованием алгоритма MD5. Хэширование производится для файлов, прошедших первичную фильтрацию по размеру и проверку MIME-сигнатур. Полученные хэш-суммы служат для быстрого группирования файлов с идентичным содержимым, что существенно ускоряет процесс обнаружения дубликатов, снижая число необходимых последующих сравнений.

md5.c – реализация алгоритма MD5. Необходим для исполнения filter_hash.c.

filter_cmp.c – модуль, отвечающий за окончательное побайтовое сравнение содержимого файлов. Если два файла, предварительно прошедшие фильтрацию по размеру, MIME-сигнатурам и хэш-суммам, остаются кандидатами на идентичность, то для окончательной проверки вызывается функция, которая открывает оба файла, считывает их данные блоками и сравнивает их по байтам. Если хотя бы один блок данных отличается, файлы считаются неодинаковыми. Данный этап применяется только к уже

отфильтрованным файлам, что минимизирует затраты вычислительных ресурсов.

`result.h` – заголовочный файл, определяющий набор структур данных и прототипов функций, предназначенных для хранения и обработки результатов работы утилиты. В `result.h` описываются структуры для представления групп дубликатов, где каждая группа содержит список файлов с одинаковым содержимым, а также дополнительные поля для хранения информации о количестве найденных экземпляров, суммарном размере и другой статистике. Благодаря единому определению типов данных и прототипов функций, остальные модули программы могут использовать этот интерфейс для передачи и вывода результатов, что упрощает интеграцию и поддержку кода.

`result.c` – реализация функций, отвечающих за обработку, группировку и форматирование полученных данных о дубликатах. Здесь производится подготовка результата для вывода пользователю. Функции из `result.c` обеспечивают четкое и понятное представление данных – как в интерактивном режиме, так и в рамках автоматизированной обработки – что позволяет пользователю легко интерпретировать информацию о найденных дубликатах и принять необходимые меры.

`deletion.c` – модуль удаления обнаруженных дубликатов. В этом модуле настроены два основных режима работы: интерактивный режим, когда программа запрашивает у пользователя подтверждение или выбор файла для сохранения из каждой группы дубликатов, и неинтерактивный режим, при котором сохраняется первый файл в группе, а остальные удаляются автоматически. Также в `deletion.c` реализована обработка ошибок, проверка допустимости операции удаления и ведение логирования, что обеспечивает надежность и безопасность выполнения критических операций. Такой подход дает возможность адаптировать работу утилиты под различные сценарии – от

тщательного контроля удалений до быстрого массового освобождения дискового пространства.

`verbose.h` – заголовочный файл, содержащий объявления макросов, констант и прототипы функций, предназначенных для режима подробного вывода (`verbose`). Благодаря ему другие модули программы могут вызывать функции логирования, которые отображают информацию о ходе выполнения программы, ошибки, предупреждения и статус выполнения различных операций. Модуль помогает отслеживать работу утилиты на каждом этапе.

`Makefile` – файл сборки, предназначенный для автоматизации компиляции проекта. В нем заданы правила и зависимости для компиляции всех исходных файлов, установлены компиляторные флаги, а также цели для сборки исполняемого файла, очистки временных файлов (`clean`) и, возможно, дополнительные команды для тестирования. `Makefile` упрощает процесс сборки проекта и позволяет разработчикам быстро компилировать программу на различных платформах.

`README.md` – документация проекта в формате `Markdown`, содержащая общее описание утилиты, её цели, особенности и инструкции по установке и использованию. В этом файле изложены ключевые требования, примеры настройки параметров, способы запуска программы и общая информация для пользователей и разработчиков. `README.md` служит первым источником информации о проекте, облегчая понимание его функционала и идеи.

`bdupes.1` – файл страницы руководства (`man page`) для утилиты, оформленный в формате `troff`. Здесь подробно описаны назначение программы, поддерживаемые опции, примеры использования, информация об авторе и лицензии. Данный файл позволяет пользователям получить быстрый доступ к справочной информации через команду `man`, обеспечивая удобное и систематизированное представление возможностей утилиты.

`examples` – директория, содержащая демонстрационные файлы, на которых можно тестировать работу утилиты в различных сценариях. Содержимое этой папки служит хорошей основой для проведения тестов и примера интеграции утилиты в реальные задачи.

3.2 Взаимодействие модулей

Взаимодействие модулей организовано через четко определённые интерфейсы, предоставляемые соответствующими заголовочными файлами, что обеспечивает единообразие обмена данными между компонентами и позволяет каждому модулю выполнять свою специализированную задачу.

Главный модуль программы (`main.c`) выполняет функции инициализации и координации работы утилиты. Он начинает с вызова функции `parse_arguments`(), реализованной в `args.c` и объявленной в `args.h`, которая анализирует параметры командной строки и заполняет структуру настроек. Полученные настройки сохраняются либо в глобальных переменных, либо передаются через специальные структуры, что гарантирует корректное использование конфигурационных данных всеми остальными модулями.

После разбора аргументов `main.c` вызывает модуль `selection.c` для сканирования указанных директорий. В рамках своей работы `selection.c` использует системные вызовы для обхода файловой системы, собирает информацию о файлах и применяет фильтрацию по флагам `-G` и `-L`. Таким образом, уже на этапе сканирования исключаются файлы, не удовлетворяющие заданным критериям, что уменьшает объем данных для дальнейшего анализа.

Далее полученный список файлов передается на последовательную фильтрацию. Сначала модуль `filter_size.c` исключает файлы, не имеющие схожих размеров с другими файлами. Модуль `filter_mime.c` анализирует

первые байты каждого файла для определения его MIME-сигнатуры, что позволяет отсеять файлы с различным типом содержимого. Затем те файлы, которые успешно проходят проверку по MIME, обрабатываются в модуле filter_hash.c, где вычисляются хэш-суммы с использованием алгоритмов, таких как MD5 или SHA-256. Полученные хэш-суммы позволяют быстро сгруппировать файлы с идентичным содержимым. Если возникает необходимость в окончательном подтверждении совпадения, модуль filter_cmp.c проводит побайтовое сравнение содержимого файлов, гарантируя абсолютную точность идентификации дубликатов.

Результаты всех этапов фильтрации объединяются в модуле result.c, который, используя структуры и прототипы, определенные в result.h, формирует итоговые группы дубликатов. Эти структурированные данные затем доступны для дальнейших операций, например, для представления пользователю или для автоматизированного удаления.

Модуль deletion.c получает сформированные группы дубликатов и осуществляет удаление лишних файлов. Здесь реализована логика работы в интерактивном и неинтерактивном режимах, при этом необходимые политики и настройки, полученные на предыдущих этапах, используются для обеспечения безопасности операций удаления.

Наконец, модуль, определённый в verbose.h, интегрируется с каждым из упомянутых компонентов, обеспечивая вывод подробной информации о ходе выполнения программы.

Таким образом, взаимодействие модулей в программе происходит по цепочке, где данные последовательно проходят через этапы парсинга, отбора, фильтрации, группировки и окончательной обработки. Каждый модуль выполняет свою задачу, а обмен информацией между ними осуществляется через стандартизированные интерфейсы, что позволяет добиться высокой модульности, расширяемости и стабильности работы утилиты.

4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

4.1 Сканирование директории

Модуль сканирования директории разбит на несколько функций и основана на использовании стандартных системных вызовов для работы с файловой системой.

Сначала определены глобальные переменные, в том числе указатель на динамический массив `file_list`, счетчик `file_count` и его емкость `file_list_capacity`. Эти переменные используются для хранения информации о каждом найденном файле. Структура `file_entry` содержит путь к файлу и размер файла.

Функция `add_file_entry` отвечает за добавление нового файла в глобальный список. Если количество записей достигло текущей емкости массива, происходит перераспределение памяти с увеличением емкости. Затем путь к файлу копируется в выделенный блок памяти с обеспечением корректного завершения строки, и размер файла сохраняется. После этого счетчик `file_count` увеличивается.

Функция `scan_directory` реализует рекурсивное сканирование заданного каталога. В ней открывается указанная директория через `opendir`. Если директория не открылась, выводится сообщение об ошибке. Далее с помощью `readdir` происходит перебор содержимого директории. Специальные записи `"."` и `".."` пропускаются. Для каждого найденного элемента формируется полный путь, используя `snprintf` с учетом наличия завершающего символа '/' в исходном пути.

После получения полного пути вызывается системный вызов `stat`, чтобы получить информацию о файле или каталоге. Если `stat` завершается ошибкой, выводится сообщение об ошибке с указанием проблемного пути, и обработка продолжается.

Если объект является каталогом и установлен флаг рекурсии (recursive_flag), то функция сканирования вызывается рекурсивно для данного подкаталога. Если же объект является обычным файлом, вызывается функция add_file_entry для добавления информации о файле в глобальный массив.

Функция filter_size_list используется для фильтрации списка файлов по размеру. Она проверяет, заданы ли ограничения по минимальному (min_size) или максимальному (max_size) размеру. Если ни одно из ограничений не установлено (значения равны нулю), функция сразу возвращается без действий. Затем происходит проход по всем элементам массива file_list. Для каждого файла проверяется, удовлетворяет ли его размер ограничениям: если размер меньше минимального или больше максимального, файл пропускается. Массив перезаписывается так, чтобы в его начале оказались только удовлетворяющие условиям элементы. После завершения прохода счетчик file_count обновляется значением j, что соответствует количеству оставшихся файлов.

4.2 Фильтрация списка кандидатов

Модуль filter_size.c реализует фильтрацию списка файлов, оставляя в итоговом наборе только те файлы, у которых обнаружены как минимум один дубликат по размеру, то есть файлы, для которых имеется группа с более чем одним элементом. Схема алгоритма приведена в приложении А.

Сначала производится сортировка глобального массива file_list с использованием функции qsort. Функция compare_file_entry сравнивает два элемента структуры file_entry по значению поля file_size, что позволяет расположить файлы в порядке возрастания их размера. Такая сортировка гарантирует, что файлы с одинаковым размером будут стоять подряд в массиве.

После сортировки начинается проход по отсортированному массиву. Переменная `i` используется как начало текущей группы файлов с одинаковым размером. С помощью цикла `while` определяется граница группы: переменная `j` увеличивается, пока `file_list[j].file_size` равен `file_list[i].file_size`. Таким образом, файлы от индекса `i` до `j-1` образуют группу кандидатов для дальнейшей обработки.

Если размер группы оказывается больше единицы, то есть найдено более одного файла с данным размером, все элементы этой группы копируются в новый динамический массив `filtered_list`. При этом осуществляется динамическое выделение памяти для `filtered_list` с увеличением емкости по мере необходимости. Если же в группе содержится ровно один файл, он не копируется в новый массив, а его путь выводится функцией `verbose_log_path` – это сигнализирует о том, что для данного размера файл является уникальным и не имеет дубликатов.

По завершении прохода по всему массиву производится освобождение памяти исходного файла `file_list`, после чего указатель `file_list` переназначается на новый массив `filtered_list`. Также обновляется переменная `file_count` до значения `filtered_count`, которое отражает количество файлов, удовлетворяющих условию наличия дубликатов по размеру.

Модуль `filter_mime.c` фильтрует список файлов, оставляя только те, у которых в группе с одинаковым размером встречается `MIME`-тип более одного раза. Схема алгоритма приведена в приложении Б. Алгоритм работает следующим образом.

Сначала проверяется, что список файлов не пуст. Затем инициализируется библиотека `libmagic` – открывается дескриптор посредством `magic_open` с флагом `MAGIC_MIME` и загружается база `MIME`-

типов с помощью magic_load. Если инициализация завершается неудачей, программа выводит сообщение об ошибке и завершает работу.

Далее модуль начинает проход по глобальному массиву file_list. Цикл проходит по файлам, группируя подряд идущие элементы с одинаковым значением file_size. Для каждой такой группы определяется её размер, после чего выделяется временный массив (mime_array) для хранения MIME-типов, по одному для каждого файла группы.

Для каждого файла группы вызывается функция magic_file, которая на основании содержимого файла возвращает строку с MIME-типом. Эта строка дублируется с помощью strdup и сохраняется в массиве mime_array, чтобы не зависеть от внутреннего буфера libmagic. При возникновении ошибок выделения памяти производится корректная очистка и завершение работы.

После того, как для всех файлов группы получены MIME-типы, выполняется их сравнительный анализ. Для каждого файла (индекс k в группе) подсчитывается, сколько раз его MIME-тип встречается в той же группе. Если MIME-тип встречается более одного раза, значит, имеются хотя бы два файла одного размера с одинаковой структурой содержимого – такой файл добавляется в новый динамический массив, в который будут собраны все кандидаты для дальнейшей проверки. Если MIME-тип уникален для данной группы, файл не добавляется в итоговый список, а его путь выводится через функцию логирования как уникальный (это помогает проинформировать пользователя, что для данного размера подобных файлов не найдено).

После обработки всех файлов в группе временный массив mime_array очищается – все его элементы освобождаются, а затем сам массив освобождается. Затем индекс i сдвигается к следующей группе (устанавливается i = j) и весь процесс повторяется для следующих файлов в file_list.

После того как все группы обработаны, производится освобождение исходного массива file_list, и указатель на него переназначается на новый массив, содержащий только те файлы, которые прошли фильтрацию по MIME-типа. Также обновляется значение file_count, отражающее число оставшихся кандидатов. В конце работы дескриптор libmagic закрывается.

Модуль filter_hash.c реализует фильтрацию списка файлов на основе вычисления MD5-хешей для обнаружения дубликатов. Схема работы алгоритма приведена в приложении В.

Сначала определяется структура file_entry (содержащая полный путь и размер файла), которая используется для хранения информации о файле, и вспомогательная структура file_hash, объединяющая file_entry с указателем на строку с MD5-хешем. Глобальные переменные file_list, file_count и file_list_capacity, определённые в selection.c, представляют исходный список файлов для анализа.

Функция compute_md5 открывает указанный файл в бинарном режиме и создает MD5-контекст. Затем файл читается блоками по 4096 байт, при этом каждый блок передается функции MD5_Update для обновления хеш-состояния. По завершении чтения вызывается MD5_Final, чтобы получить итоговый 16-байтовый хеш, который затем преобразуется в строку длиной 32 символа в шестнадцатеричном представлении. Эта строка выделяется динамически и возвращается вызывающему коду; в случае ошибок возвращается NULL, или же присваивается пустая строка, чтобы исключить файл из дальнейшей обработки. Функция compare_file_hash используется с qsort для сортировки массива file_hash по значению MD5-хеша.

В основной функции модуля, filter_hash_list, сначала проверяется наличие файлов. Далее выделяется массив hash_array размером file_count, где для каждого файла из file_list сохраняется копия самой записи и вычисленный для него MD5-хеш. Если вычисление MD5 по какой-либо причине

не удалось, вместо него присваивается пустая строка, чтобы файл впоследствии не вошел в группу дубликатов.

После вычисления хешей весь массив hash_array сортируется по MD5-хешам. Затем производится итерация по отсортированному массиву: для каждой группы подряд идущих элементов с одинаковым хешем определяется размер группы. Если группа содержит более одного файла – это указывает на наличие дубликатов – все файлы из этой группы добавляются в новый массив filtered_list посредством динамического перераспределения памяти. Если же группа состоит из единственного элемента, путь такого файла выводится через функцию логирования (verbose_log_path), и считается, что для данного файла дубликатов не найдено.

После обработки всех групп временный массив hash_array и все строки, содержащие MD5-хеши, освобождаются. Исходный список file_list освобождается, после чего указатель file_list заменяется на новый отфильтрованный список filtered_list, а переменные file_count и file_list_capacity обновляются в соответствии с числом оставшихся файлов.

Модуль filter_cmp.c производит побайтовое сравнение кандидатов. Для этого используется функция files_are_identical, которая открывает два файла в бинарном режиме, читает их блоками по 4096 байт и сравнивает содержимое с помощью функции memcmp. Если при чтении обнаруживается несоответствие по количеству прочитанных байтов или содержимого блока, файлы считаются различными; если же чтение завершается без ошибок и все блоки совпадают, функция возвращает значение 1, указывающее на идентичность файлов.

Основная функция модуля filter_cmp_list последовательно перебирает все файлы из глобального списка file_list. Сначала выделяется временный массив-флаг keep_flags, размер которого соответствует

количеству файлов. После этого осуществляется двойной цикл: для каждого файла с индексом `i` сравнивается каждый последующий файл с индексом `j`, при условии что их размеры равны. Если два файла оказались идентичными по содержимому (функция `files_are_identical` возвращает 1), соответствующие элементы массива `keep_flags` для обоих файлов устанавливаются в 1, что означает, что данный файл имеет дубликат.

После завершения попарных сравнений формируется новый динамический массив `filtered_list`, куда копируются только те записи, для которых соответствующий флаг `keep_flags` равен 1. Файлы, не имеющие хотя бы одного совпадения с другими, считаются уникальными, и их пути выводятся через функцию логирования. Если ни один файл не был найден с дубликатом, выводится сообщение «не найдены».

В завершении временный массив `keep_flags` освобождается, исходный список `file_list` также освобождается, а указатель `file_list` переназначается на новый массив `filtered_list`. При этом значение `file_count` обновляется до количества оставшихся файлов, удовлетворяющих условию идентичности. Таким образом, модуль оставляет в итоговом списке только те файлы, для которых подтверждена полная совпадающая копия путем тщательного побайтового сравнения.

4.3 Вывод списка дубликатов и их удаление

Модуль `result` отвечает за вывод итоговых результатов обработки списка файлов, которые были обнаружены как дубликаты.

Функция `print_filtered_file_list` начинается с проверки наличия файлов в глобальной переменной `file_list` (если `file_count` равен нулю, выводится сообщение о том, что файлы-дубликаты не найдены и функция завершается). Функция проходит по списку с использованием цикла `while`, группируя подряд идущие записи с одинаковым значением `file_size`. Для

каждой группы сначала определяется текущий размер файлов, затем перебираются элементы группы. Если флаг `time_flag` включен, для каждого файла с помощью вызова функции `stat` получается информация о последнем времени модификации, которое затем форматируется через `localtime` и `strftime` в строку вида «ГГГГ-ММ-ДД ЧЧ:ММ:СС» и выводится вместе с путем к файлу. Если же время не требуется, выводится только путь к файлу. Между группами файлов выводится пустая строка, что позволяет визуально разделить записи.

Функция `print_summary` предназначена для расчёта и вывода сводной статистики по найденным дубликатам при наличии флага `-m`. Алгоритм проходит по всему списку `file_list`, группированному по одинаковому размеру файлов. Для каждой группы вычисляется количество записей, а затем для каждого файла внутри группы вызывается `stat` для определения реального размера на диске (определяемого произведением количества выделенных блоков на 512 байт). Для группы рассчитывается суммарный размер, при этом фиксируется максимальный выделенный размер (предполагается, что файл с наибольшим выделенным размером остаётся оригиналом, а остальные могут быть удалены). Если в группе более одного файла, число групп-дубликатов увеличивается, а также суммируется количество файлов-дубликатов (группа из `n` файлов приводит к тому, что `n-1` файл можно удалить) и суммарный размер потерь (`total_waste`) вычисляется как разница между общим размером группы и максимальным размером выделенного блока. В конце функция переводит суммарный объём потерь из байт в мегабайты и выводит сообщение с числом дубликатов, числом групп и общим объёмом «ненужного» места на диске.

Функция `print_size_listing` выводит список файлов, сгруппированных по их логическому размеру при наличии флага `-S`. Функция проходится по всему массиву `file_list`, для каждой группы файлов с

одинаковым размером печатая сначала заголовок, где указывается общий размер для группы, а затем последовательно выводит пути для каждого файла из этой группы. Это позволяет пользователю увидеть, какие файлы имеют одинаковый размер.

Функция `print_size_time_listing` аналогична предыдущей, но для каждой записи дополнительно выводит информацию о времени последней модификации файла. Затем вместе с путем к файлу эта информация выводится на экран. Как и в других случаях, группы разделены пустой строкой для удобства восприятия.

Модуль `deletion.c` реализует два режима удаления обнаруженных дубликатов файлов – интерактивный и неинтерактивный. Оба режима работают с глобальным массивом `file_list`, который был сформирован на предыдущих этапах обработки.

В интерактивном режиме (функция `interactive_delete_duplicates`) сначала проверяется, что список файлов не пуст. Затем модуль проходит по всему списку `file_list`, группируя подряд идущие записи по одинаковому значению `file_size` – считается, что файлы с одинаковым размером являются дубликатами. В первой части цикла производится подсчёт общего числа наборов дубликатов (групп, где количество файлов в группе не меньше двух). Это значение используется для информирования пользователя при дальнейшем взаимодействии.

После этого цикл повторно проходит по списку. Для каждой группы файлов с одинаковым размером выводится нумерованный список, в котором для каждого файла отображается его путь. Если флаг `time_flag` установлен, дополнительно показывается время последней модификации файла; для этого с помощью функции `stat` извлекается информация о файле, затем преобразуется время модификации через `localtime` и `strftime` в форматированную строку (например, «`YYYY-MM-DD HH:MM`»). Для каждой

группы пользователь получает приглашение – сообщение с указанием номера текущей группы из общего количества, а также информацией о том, какие номера доступны для выбора (от 1 до `N`, где `N` – число файлов в группе). Приглашение предлагает ввести номер файла, который будет сохранён (остальные будут удалены), либо строку «`all`» для того, чтобы сохранить всю группу, либо «`quit`» для выхода из режима удаления. Если ввод пользователя некорректен, приглашение повторяется до получения допустимого значения.

Если пользователь выбирает «`all`», файлы группы остаются без изменений. Если выбран номер, модуль сохраняет выбранный файл и для остальных файлов в группе вызывается функция `remove()` для их удаления, после чего выводится сообщение об удалении или сообщение об ошибке, если удаление не удалось.

В неинтерактивном режиме (функция `non_interactive_delete_duplicates`) обработка групп происходит автоматически без запроса ввода от пользователя. Для каждой группы файлов с одинаковым размером, если группа содержит больше одного файла, первый файл считается оригиналом и сохраняется, а для всех остальных файлов группы вызывается функция `remove`(), чтобы удалить их. При этом для оригинала выводится сообщение с положительным индикатором `[+]`, а для удалённых файлов – отрицательный индикатор `[-]`.

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

5.1 Установка и сборка

1. Перед началом работы убедитесь, что в системе установлены необходимые зависимости. Для Fedora выполните команду:

```
sudo dnf install libmagic-devel
```

2. Сборка проекта производится с использованием Makefile. Для этого в командной строке в каталоге с исходным кодом выполните команду:

```
make
```

В результате будет скомпилирован исполняемый файл, например, с именем bdupes.

3. Для обеспечения удобного доступа к справке скопируйте страницу руководства в системный каталог с помощью команд:

```
sudo cp bdupes.1 /usr/share/man/man1/
sudo mandb
```

4. Установите готовую утилиту, переместив исполняемый файл в системный каталог, доступный вашему пользователю. Обычно это каталог /usr/local/bin:

```
sudo cp bdupes /usr/local/bin/
```

После проделанных шагов утилита готова к работе. Запустить её можно командой:

```
bdupes [options] <path>
```

При необходимости просмотрите справочную информацию с помощью страницы man:

```
man bdupes
```

5.2 Синтаксис

Команда запуска утилиты имеет следующий синтаксис:

`bdupes [ -r ] [ -t ] [ -m ] [ -S ] [ -v ] [ -d ] [ -n ] [ -G размер ] [ -L размер ] <каталог>`

`-r` – включает рекурсивное сканирование; утилита будет обходить не только указанный каталог, но и все его подкаталоги.

`-t` – выводит время последней модификации для каждого файла. При включенном флаге в списке отображается дата и время изменения файла.

`-m` – обеспечивает вывод сводной информации по наборам дубликатов: количество групп дубликатов и общий объём занимаемого места.

`-S` – выводит размеры файлов, позволяя увидеть логический размер каждого файла при отображении дублей.

`-v` – включает подробный режим, при котором выводятся дополнительные детали выполнения утилиты.

`-d` – активирует режим удаления дубликатов; после поиска дубликатов утилита переходит к их удалению.

`-n` – используется вместе с опцией `-d` и включает неинтерактивный режим удаления, в котором из каждой группы сохраняется первый файл, а все остальные удаляются без запроса подтверждения.

`-G размер` – устанавливает минимальный размер файла, который будет учитываться для обработки. Файлы, размер которых меньше указанного значения, игнорируются.

`-L размер` – устанавливает максимальный размер файла для обработки. Файлы, превышающие указанное значение, не будут учитываться.

`<каталог>` – путь к каталогу, в которой будет проводиться поиск дубликатов.

5.3 Примеры использования

Ниже приведены несколько типичных примеров использования утилиты, которые помогут быстро разобраться с функционалом и выбрать нужный режим работы.

1. Интерактивное удаление с рекурсивным сканированием и выводом времени изменений:

```
bdupes -rtd пример_каталога
```

запускает утилиту в режиме рекурсивного сканирования подкаталогов (`-r`), выводит время последней модификации файлов (`-t`) и активирует интерактивный режим удаления дубликатов (`-d`). После обнаружения групп дубликатов отображается нумерованный список файлов каждой группы, где предложено выбрать, какой файл сохранить, а какие удалить.

2. Неинтерактивное удаление с ограничением по размеру файлов:

```
bdupes –rdn –G 1024 -L1048576 пример_каталога
```

выполняет рекурсивное сканирование (`-r`), включает режим удаления дубликатов (`-d`) в неинтерактивном режиме (`-n` – сохраняется только первый файл в каждой группе), а также устанавливает минимальное (`-G 1024` – 1024 байта) и максимальное (`-L 1048576` – 1 мегабайт) ограничение по размеру файлов. То есть, будут обработаны только файлы размером от 1КБ до 1МБ, а из каждой группы автоматически сохранится первый файл, остальные будут удалены без дополнительных запросов.

3. Поиск дубликатов без удаления с подробным выводом:

```
bdupes -rv пример_каталога
```

запускает утилиту в рекурсивном режиме, выводит подробный лог выполнения (`-v`) и только ищет дубликаты без активации режима удаления. Это полезно для предварительного анализа и оценки объема дублирующихся данных перед удалением.

4. Получение сводной информацию по найденным дубликатам после сканирования.

```
bdupes -m пример_каталога
```

утилита не выводит подробный список всех файлов, а вместо этого подсчитывает и отображает итоговые параметры: количество групп дубликатов, число файлов-дубликатов и суммарный объём занимаемого дискового пространства, который может быть освобождён.

По завершении работы утилита выводит итоговое сообщение:

3 дублирующихся файлов (в 2 наборах), занимают 17.0 мегабайт

5. Получение детального представление по каждому дубликату с указанием времени последней модификации и размера файлов.

```
bdupes -tS пример_каталога
```

Результат позволяет пользователю сопоставить, какие файлы являются последними по времени и/или каким логическим размером располагаются дубликаты. По завершении работы утилита выводит итоговое сообщение:

```
255773 байт в каждом:
2025-05-08 12:20 ./examples/folder1/folder2/folder3/xml_copy2.xml
2025-05-08 12:20 ./examples/folder1/folder2/xml.xml
2025-05-08 12:20 ./examples/folder1/folder2/xml_copy.xml

17303073 байт в каждом:
2025-05-08 12:20 ./examples/folder1/image.png
2025-05-08 12:20 ./examples/folder1/image_copy.png
```

Для получения полного списка опций и подробностей по их применению необходимо открыть справочную страницу командой:

```
man bdupes
```

Результаты работы программы в различных сценариях приведены в приложении Г.

ЗАКЛЮЧЕНИЕ

В результате работы была разработана программа-аналог `fdupes` с ограничением поиска и налазиа по сигнатурам и `MIME`-типам файлов. Утилита представляет собой комплексное решение для поиска и удаления дубликатов файлов, которое сочетает в себе высокую точность и удобство использования. В основе утилиты лежит многоступенчатый алгоритм, включающий рекурсивное сканирование директорий, первичную фильтрацию по размерам, последующее уточнение с помощью анализа `MIME`-типа, вычисления `MD5`-хешей и завершающее побайтовое сравнение для окончательной проверки идентичности файлов. Такая последовательность обработок позволяет эффективно отсеивать нерелевантные данные, минимизируя нагрузку на систему и сокращая время поиска, что особенно важно при работе с большим количеством файлов.

Модульная архитектура проекта обеспечивает четкое разделение функциональных компонентов. Это упрощает поддержку и тестирование утилиты, а также открывает возможности для ее дальнейшего расширения и оптимизации, например, за счет внедрения дополнительных критериев фильтрации или улучшения алгоритмов сканирования.

Подробное руководство пользователя и страница справки, оформленная в формате `man`, делают применение утилиты доступным даже для тех, кто не обладает глубокими техническими знаниями.

Программа успешно решает задачу по выявлению дублирующихся файлов, позволяя оптимизировать использование дискового пространства и облегчая управление данными как для системных администраторов, так и для обычных пользователей. Практическая эффективность, гибкость в настройке и перспективы дальнейшего развития делают этот проект ценным инструментом для работы с большими массивами файловых данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] libmagic – File Type Determination Library // Darwinsys. - URL: http://www.darwinsys.com/file/ (дата обращения: 15.01.2025).

[2] OpenSSL: Cryptography and SSL/TLS Toolkit Documentation // OpenSSL Project. - URL: https://www.openssl.org/docs/ (дата обращения: 22.01.2025).

[3] GNU Make Manual // GNU Project. - URL: https://www.gnu.org/software/make/manual/ (дата обращения: 13.01.2025).

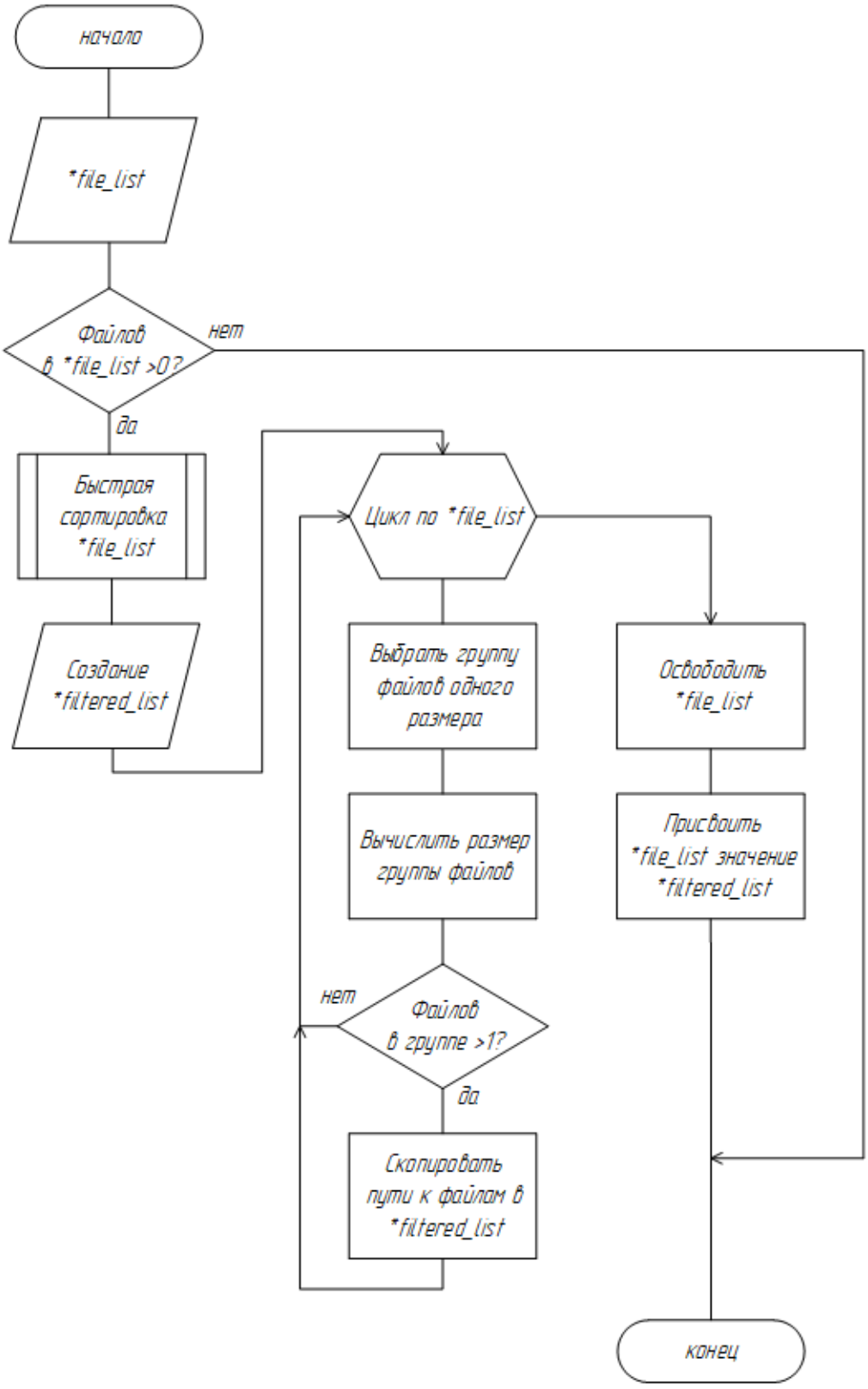
[4] fdupes: Duplicate File Finder // GitHub. - URL: https://github.com/adrianlopezroche/fdupes (дата обращения: 10.01.2025).

[5] System Interfaces and API Documentation // Man7.org. - URL: http://man7.org/linux/man-pages/ (дата обращения: 19.01.2025).

[6] Тейнсли, Д. Linux и Unix программирование в Shell. Руководство разработчика: Пер. с англ. – К.: Издательская группа BHV, 2001 – 464 с.

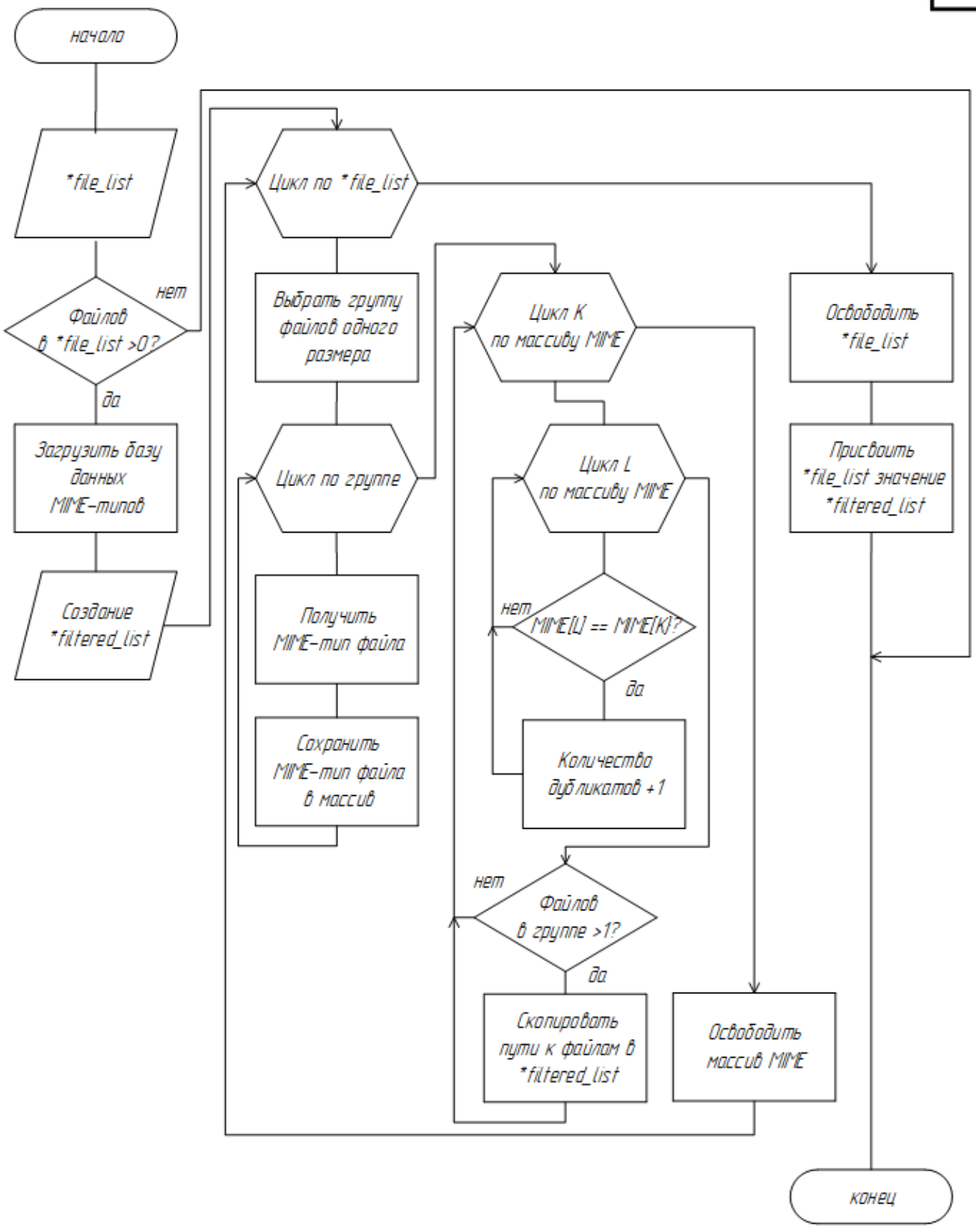
[7] Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. — СПб.: Питер, 2015. — 1120 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Схема алгоритма сравнения файлов по размеру



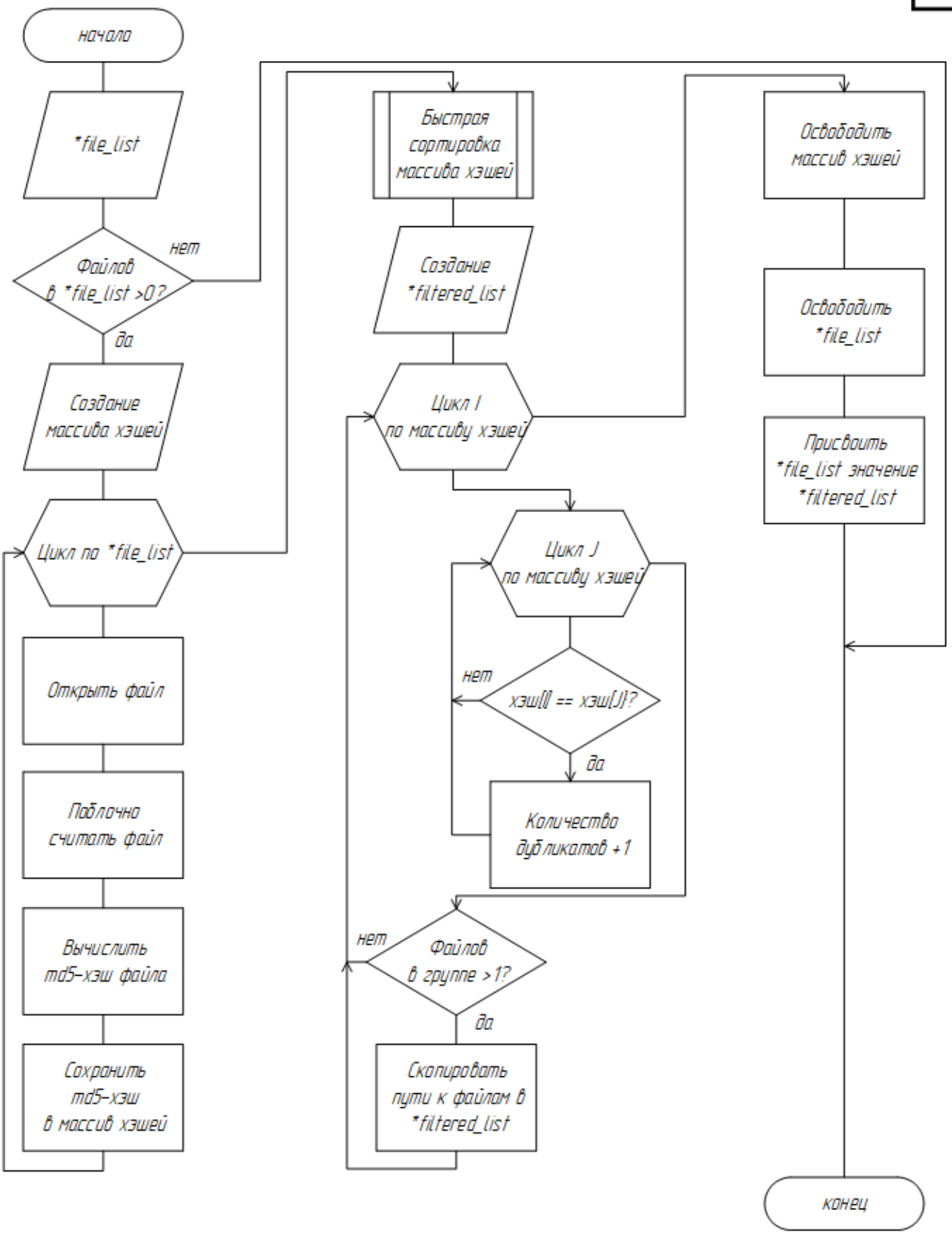
Изм	Л.	№ докум.	Подп.	Дата	ГУИР.400201.001		
Разраб.		Бабрик			Программа-аналог fdupes с ограничением поиска и анализа по сигнатурам и MIME-типам файлов Схема алгоритма сравнения файлов по размеру	Лит.	Лист 1 / Листов 1
Пров.		Поденок					БГУИР, гр.250541

ПРИЛОЖЕНИЕ Б
(обязательное)
Схема алгоритма сравнения файлов по сигнатурам и MIME-типу



Изм	Л.	№ докум.	Подп.	Дата	ГУИР.400201.001		
Разраб.		Бабрик			Программа-аналог fdupes с ограничением поиска и анализа по сигнатурам и MIME-типам файлов Схема алгоритма сравнения файлов по сигнатурам и MIME-типу	Лит.	Лист 1 / Листов 1
Пров.		Поденок					БГУИР, гр.250541

ПРИЛОЖЕНИЕ В
(обязательное)
Схема алгоритма сравнения файлов по md5-хэшу



Изм	Л.	№ докум.	Подп.	Дата	ГУИР.400201.001		
Разраб.		Бабрик			Программа-аналог fdupes с ограничением поиска и анализа по сигнатурам и MIME-типам файлов Схема алгоритма сравнения файлов по md5-хэшу	Лит.	Лист 1 / Листов 1
Пров.		Поденок					БГУИР, гр.250541

ПРИЛОЖЕНИЕ Г
(обязательное)
Результаты работы программы

1) Получение списка дубликатов с временем последнего изменения и размером дубликатов.

```
$ ./bdupes -rSt examples/
255773 байт в каждом:
2025-05-08 12:20 examples/folder1/folder2/folder3/xml_copy2.xml
2025-05-08 12:20 examples/folder1/folder2/xml.xml
2025-05-08 12:20 examples/folder1/folder2/xml_copy.xml

17303073 байт в каждом:
2025-05-08 12:20 examples/folder1/image.png
2025-05-08 12:20 examples/folder1/image_copy.png
```

2) Получение сводной информации о дубликатах флагом –m. Остальные флаги игнорируются, кроме рекурсии.

```
$ ./bdupes -rmtS examples/
3 дублирующихся файлов (в 2 наборах), занимают 17.0 мегабайт
```

3) Получение списка дубликатов в папке без учета вложенных папок. В корневой папке заведомо нет дубликатов.

```
$ ./bdupes examples/
Файлы-дубликаты не найдены
```

4) Интерактивное удаление дубликатов с временем последнего изменения файла.

```
$ ./bdupes -rdt examples2/
[1] [2025-05-12 15:28]
examples2/folder1/folder2/folder3/xml_copy2.xml
[2] [2025-05-12 15:28] examples2/folder1/folder2/xml.xml
```

```
[3] [2025-05-12 15:28] examples2/folder1/folder2/xml_copy.xml

Набор 1 из 2, сохранить файл [1 - 3, all, quit]: 2
Удален: examples2/folder1/folder2/folder3/xml_copy2.xml
Удален: examples2/folder1/folder2/xml_copy.xml

[1] [2025-05-12 15:28] examples2/folder1/image.png
[2] [2025-05-12 15:28] examples2/folder1/image_copy.png

Набор 2 из 2, сохранить файл [1 - 2, all, quit]: 1
Удален: examples2/folder1/image_copy.png
```

5) Неинтерактивное удаление дубликатов.

```
$ ./bdupes -rnd examples
   [+] examples/folder1/folder2/folder3/xml_copy2.xml
   [-] examples/folder1/folder2/xml.xml
   [-] examples/folder1/folder2/xml_copy.xml

   [+] examples/folder1/image.png
   [-] examples/folder1/image_copy.png
```

6) Неверное указание пути.

```
$ ./bdupes -rnd
Ошибка: не указан начальный путь
```

7) Неинтерактивное удаление файлов размером более 1Мб.

```
$ ./bdupes -rdn -G $((1024*1024)) examples2/
   [+] examples2/folder1/image.png
   [-] examples2/folder1/image_copy.png
```

8) Просмотр списка дубликатов в режиме подробного вывода.

```
$ ./bdupes -rv  examples2/
[verbose] Сканирование директории:
```

```
[verbose] examples2/
[verbose] Найден каталог:
[verbose] examples2/folder1
[verbose] Сканирование директории:
[verbose] examples2/folder1
[verbose] Найден каталог:
[verbose] examples2/folder1/folder2
[verbose] Сканирование директории:
[verbose] examples2/folder1/folder2
[verbose] Найден каталог:
[verbose] examples2/folder1/folder2/folder3
[verbose] Сканирование директории:
[verbose] examples2/folder1/folder2/folder3
[verbose] Фильтрация по объему памяти...
[verbose] Уникальные файлы:
[verbose] examples2/table.csv
[verbose] examples2/folder1/folder2/pdf.pdf
[verbose] examples2/archive.tar.gz
[verbose] examples2/video.mp4
[verbose] examples2/music.wav
[verbose] Фильтрация по типу файла...
[verbose] Уникальные файлы:
[verbose] examples2/folder1/image.png
[verbose] Фильтрация по хешу...
[verbose] Уникальные файлы:
[verbose] examples2/folder1/image.png
[verbose] examples2/folder1/image_copy_2.png
[verbose] Фильтрация побайтовым сравнением...
[verbose] Уникальные файлы:
[verbose] не найдены
examples2/folder1/folder2/xml.xml
examples2/folder1/folder2/xml_copy.xml
examples2/folder1/folder2/folder3/xml_copy2.xml
```

9) Попытка одновременно получения сводной информации о дубликатах и их удаления

```
$ ./bdupes -rdm  examples2/
Флаги -m и -d несовместимы
```