

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

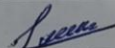
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»  
Профиль подготовки: «Программное обеспечение средств вычислительной  
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**УТИЛИТА УПРАВЛЕНИЯ КЛАСТЕРНОЙ  
ФАЙЛОВОЙ СИСТЕМОЙ**

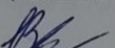
Студент

  
(подпись, дата)

Ланцов А.С.

13.06.2023

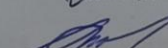
Руководитель

  
(подпись, дата)

Егоров В.Ю.

14.06.2023

Нормоконтролёр

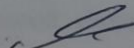
  
(подпись, дата)

Бычков А.С.

14.06.2023

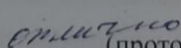
Работа допущена к защите (протокол зас. кафедры от 2.06.23 г. № 14)

Заведующий кафедрой

  
(подпись)

Митрохин М.А.

Работа защищена с отметкой

  
(подпись)

(протокол заседания ГЭК от 16.06.23 № 18)

Секретарь ГЭК

  
(подпись)

Гурин Е.И.

Пенза 2023

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет  
вычислительной техники

Кафедра  
«Вычислительная техника»

«Утверждаю»  
Заведующий кафедрой  
М.А. Митрохин  
« 26 » 04 20 23 г.

**ЗАДАНИЕ  
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Ланцов Андрей Сергеевич гр. 19ВВП2 факультета вычислительной техники
2. Тема работы: «Утилита управления кластерной файловой системой»  
Тема утверждена приказом ПГУ № 458/0 от " 26 " 04 20 23 г.
3. Руководитель бакалаврской работы Егоров В.Ю.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение «Утилита управления кластерной файловой системой». Программа должна быть интуитивно понятна пользователю и упрощать администрирование кластерной файловой системой. Приложение должно позволять пользователю производить: создание и изменение томов с требуемой отказоустойчивостью, мониторинг томов, управление репликацией, мониторинг активных подключений узлов хранения, управления, клиентов, безопасное завершение работы CFS.

# 5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной областей: изучить аналоги существующих кластерных файловых систем, выявить, что в них является утилитой управления системой, изучить актуальные способы создания консольных и сетевых приложений. Также необходимо изучить протокол взаимодействия с кластерной файловой системой Csfs.

# 6. Календарный график выполнения работы

| № п/п | Наименование этапов работы                                               | Объем работы | Срок выполнения         | Подпись руководителя |
|-------|--------------------------------------------------------------------------|--------------|-------------------------|----------------------|
| 1.    | Изучение предметной области                                              | 50           | 26.04.2023 - 29.04.2023 | BS                   |
| 2.    | Анализ требований                                                        | 10           | 30.04.2023 - 2.05.2023  | BS                   |
| 3.    | Выбор технологий, применяемых при проектировании и разработке            | 50           | 3.05.2023 - 6.05.2023   | BS                   |
| 4.    | Описание алгоритмов                                                      | 40           | 7.05.2023 - 10.05.2023  | BS                   |
| 5.    | Разработки приложения: «Утилита управления кластерной файловой системой» | 150          | 11.05.2023 - 26.05.2023 | BS                   |
| 6.    | Оформление пояснительной записки                                         | 24           | 27.05.2023 - 31.05.2023 | BS                   |

Дата выдачи задания " 26 " 04 2023 г.

Руководитель бакалаврской работы BS Егоров В.Ю.  
(подпись, дата)

Задание к исполнению принял студент BS Ланцов А.С.  
(подпись, дата)

Работу к защите допустить  
Декан факультета 16.06.2023 Фионова Л.Р.  
(подпись, дата)

## Реферат

Бакалаврская работа содержит 136 листов, 17 рисунков, 8 источников и 3 приложения

КЛАСТЕРНЫЕ ФАЙЛОВЫЕ СИСТЕМЫ, РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ, СИСТЕМНОЕ ПО, С/С++, КОНСОЛЬНЫЕ УТИЛИТЫ, TCP/IP

Цель работы – спроектировать и закодировать утилиту управления кластерной файловой системой CSFS.

Приложение должно обладать таким функционалом, как:

- создание и изменение томов с требуемой отказоустойчивостью;
- мониторинг томов;
- управление репликацией;
- мониторинг активных подключений узлов хранения, управления, клиентов;
- безопасное завершение работы CFS.

В результате проведенной работы разработано приложение, полностью соответствующее поставленной задаче. Для достижения результата были использованы следующие технологии: C++, MSVC, TCP/IP, Winsocks. Приложение протестировано и готово к использованию.

|                                                   |      |               |         |      |
|---------------------------------------------------|------|---------------|---------|------|
| <h3 style="margin: 0;">ПГУ1.090301.31.001 ПЗ</h3> |      |               |         |      |
| Изм.                                              | Лист | № докум.      | Подпись | Дата |
| Разраб.                                           |      | Ланцов А.С.   |         |      |
| Провер.                                           |      | Егоров В.Ю.   |         |      |
| Н. Контр                                          |      | Бычков А.С.   |         |      |
| Утверд.                                           |      | Митрохин М.А. |         |      |

Утилита управления кластерной файловой системой

Пояснительная записка

|                               |      |        |
|-------------------------------|------|--------|
| Лит.                          | Лист | Листов |
|                               |      | 2 136  |
| <b>ФВТ, каф. ВТ</b><br>19ВВП2 |      |        |

## Содержание

|                                                                      |     |
|----------------------------------------------------------------------|-----|
| Перечень сокращений и обозначений.....                               | 4   |
| Введение.....                                                        | 5   |
| 1 Утилита управления кластерной файловой системой.....               | 9   |
| 1.1 Анализ предметной области .....                                  | 9   |
| 1.1.1 Анализ существующих кластерных систем.....                     | 9   |
| 1.1.2 Постановка задачи на разработку .....                          | 14  |
| 1.3.1 Выбор языка программирования .....                             | 20  |
| 1.4 Проектирование и разработка .....                                | 28  |
| 1.2.1 Интерфейс .....                                                | 28  |
| 1.2.2 Общий алгоритм работы программы .....                          | 29  |
| 1.2.1 Команды .....                                                  | 29  |
| 1.2.2 Роль CommandManager.....                                       | 31  |
| 1.2.3 Парсинг аргументов командной строки.....                       | 33  |
| 1.2.4 Абстракция ключа.....                                          | 33  |
| 1.2.5 Исполнение команды .....                                       | 37  |
| 1.3 Анализ результатов .....                                         | 40  |
| 2 Охрана труда .....                                                 | 43  |
| 2.2 Основные вредоносные факторы.....                                | 43  |
| 2.3 Общие требования к организации и оборудованию рабочих мест ..... | 44  |
| 2.4 Требования к помещениям с компьютерами.....                      | 44  |
| 2.5 Правила размещения монитора.....                                 | 45  |
| 2.6 Режим труда и отдыха при использовании компьютера .....          | 46  |
| Заключение .....                                                     | 47  |
| Список используемых источников.....                                  | 48  |
| Приложение А – <i>UML</i> -диаграммы .....                           | 49  |
| Приложение Б – Листинг программы .....                               | 54  |
| Приложение В – Презентация.....                                      | 128 |

## Перечень сокращений и обозначений

ГОСТ – государственный стандарт

ПК – персональный компьютер

ФС — файловая система

ЦОД — центр обработки данных

*API* — (*Application Programming Interface*) – программный интерфейс приложения

*CFS* — (*Cluster File System*) – кластерная файловая система

*CSFS* — (*Cryptosoft Cluster File System*) — кластерная файловая система “Криптософт”.

*GUI* — (*Graphical User Interface*) — Графический интерфейс пользователя

*IDE* (*Integrated Development Environment*) — единая среда разработки

*iSCSI* — (*Internet Small Computer System Interface*) — Интерфейс небольшой

*PaaS* — (*Platform as a Service*) — платформа как услуга

*POSIX* — (*Portable Operating System Interface*) — переносимый интерфейс операционных систем

*RAID* — (*Redundant Array of Independent Disks* — избыточный массив независимых (самостоятельных) дисков)

*SaaS* — (*Software as a Service*) — программное обеспечение как услуга

*SMB* — (*Server Message Block*) — Блок сообщений сервера (сетевой протокол)

## Введение

Использование кластерных файловых систем становится все более распространенным в современных вычислительных системах благодаря их способности обеспечивать высокопроизводительные, масштабируемые и надежные решения для хранения данных. Однако управление кластерными файловыми системами может быть непростой задачей из-за комплексности и размера этих систем. В результате возникает потребность в инструментах, которые могут упростить управление и администрирование этих систем. Данная выпускная работа посвящена разработке утилиты управления кластерной файловой системой, которая предоставляет администраторам удобный интерфейс для управления и мониторинга кластерных файловых систем. Цель утилиты — упростить процесс управления кластерной файловой системой, предоставив администраторам соответствующий набор возможностей и функций.

Хотя утилиту необходимо разработать для конкретной кластерной файловой системы, ее модульная и расширяемая архитектура позволит в будущем адаптировать ее к другим кластерным файловым системам с соответствующими изменениями.

Исследование, представленное в данной работе, включает в себя комплексный анализ проблем, возникающих при управлении кластерными файловыми системами, а также изучение современных инструментов и методов. На основе этого анализа представлена разработка утилиты управления кластерной файловой системой, а также подробное описание ее функций и возможностей.

В целом, данная выпускная работа призвана внести вклад в развитие управления кластерными файловыми системами путем предоставления надежного и удобного инструмента для управления этими системами.

Облачные вычисления — это модель предоставления широкодоступного, удобного доступа по сети к общему пулу настраиваемых вычислительных ресурсов по требованию (к таким, как сети, серверы, системы хранения данных, приложения и сервисы) — определение *National Institute of Standards and Technology*. Вычислительные ресурсы быстро выделяются и высвобождаются для организации управления и взаимодействия с поставщиком услуг с минимальными усилиями со стороны заказчика.

Эта модель имеет пять основных характеристик, три модели услуг и четыре модели развертывания. Характеристики включают: самообслуживание, универсальный доступ к сети, общий пул ресурсов, эластичность и учет потребления. Сервисные модели различаются по границе контроля потребителя над предоставляемой инфраструктурой и включают в себя:

— инфраструктура как услуга (*IaaS*) — в этом случае пользователь получает контроль над всеми уровнями программного стека над облачной платформой, а именно над виртуальными машинами, сетями, объемом пространства на системе хранения данных. В этом пользователь выступает в роли администратора операционной системы и всего, что работает поверх нее, вплоть до приложений. Примерами платформ, обеспечивающих такую модель, являются *OpenStack*, *Apache CloudStack*, *Eucalyptus* и *OpenNebula*;

— программное обеспечение как услуга (*SaaS*) — в этом случае границей контроля пользователя является само приложение. Пользователь в этом случае может даже не знать, что такое виртуальная машина или операционная система, они просто работают с приложением. Примерами таких облачных продуктов являются, например, *Google Docs*, *Office 365* или *Yandex Mail*.

Модели развертывания облачных платформ включают:

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
|      |      |          |         |      |                       | 6    |
| Изм. | Лист | № докум. | Подпись | Дата |                       |      |

— платформа как услуга (*PaaS*) — облако, построенное по этой модели, вполне может быть расположено "внутри" облака модели *IaaS*. В этом случае граница пользовательского контроля проходит на уровне платформы для создания приложений, например, сервера приложений, библиотек, среды разработки или базы данных. Пользователь не контролирует и не управляет виртуальными машинами и установленными на них операционными системами, системами хранения данных и сетями. Примерами облачных *PaaS*-платформ являются *Apache Stratos*, *Cloud Foundry*, *Deis* и *OpenShift Origin*;

— частное облако — вся инфраструктура развернута в центре обработки данных (ЦОД) и служит подразделением одной компании или группы компаний;

— публичное облако — любая компания или даже частное лицо может заказать облачные услуги. Это модель развертывания, на которой зарабатывают провайдеры облачных услуги;

— облако сообщества, или общественное облако. Модель, в которой потребитель представляет собой сообщество потребителей организаций с общими целями (например, миссия, требования к безопасности, политики и соблюдение различных требований);

— гибридное облако — это комбинация двух или трех облаков, описанных выше, где различные рабочие нагрузки могут размещаться как в частном, так и в публичном или общественном облаке. Как правило, гибридное облако – это больше, чем просто сумма облаков, поскольку ему требуются механизмы и инструменты централизованного управления, распределения и миграции нагрузки между облачными инфраструктурами.

Традиционные приложения, например, база данных или почтовый сервер, требуют функционала «системы промышленной виртуализации». Этот функционал включает в себя, но не ограничивается средствами обеспечения высокой доступности, живой миграции, резервного

копирования, возможность добавлять в виртуальную машину ресурсы или забирать их. Жизненный цикл таких виртуальных машин – как правило, годы.

Современные облачные приложения – это кластер виртуальных машин. Они масштабируются горизонтально, добавлением виртуальных машин. Приложение пишется таким образом, что каждая виртуальная машина сама по себе не является критичной для функционирования всего приложения и не требует высокой доступности. Также вполне возможно, что виртуальные машины работают без сохранения состояния, что не требует их резервного копирования. Жизненный цикл подобных виртуальных машин – как правило, месяцы [1].

Сетевая файловая система – представляет собой протокол распределенной файловой системы, позволяющий пользователю на клиентском компьютере получать доступ к файлам компьютерной сети аналогично доступу к локальному хранилищу (*SMB*).

Под распределенной ФС понимается файловая система, которая предоставляет совместный доступ к файлам всем пользователям сети.

Кластерная ФС – файловая система, которая совместно используется при одновременном монтировании на нескольких серверах. Кластерные файловые системы могут предоставлять функции адресации, независимой от местоположения, и избыточность хранимых данных, которые повышают надежность хранения.

В общем случае кластерная и распределенная файловая система выполняют аналогичную функцию предоставления доступа к файлам, однако в отличие от распределенной ФС, кластерная ФС требует процедуры монтирования на конкретное устройство. Таким образом, кластерная файловая система является распределенной, но распределенная ФС может быть кластерной.

# 1 Утилита управления кластерной файловой системой

## 1.1 Анализ предметной области

### 1.1.1 Анализ существующих кластерных систем

В работе рассматриваются уже существующие решения *CFS*, проведен анализ их архитектуры, определен компонент, который служит утилитой управления этой файловой системой, выявлены их преимущества и недостатки.

*Lustre*.

*Lustre* — это популярная параллельная распределенная файловая система с открытым исходным кодом, широко используемая в средах высокопроизводительных вычислений и больших данных. Она обеспечивает высокоскоростной доступ к большим массивам данных путем распределения данных между несколькими серверами хранения и клиентами в кластере. Архитектура *Lustre* состоит из трех основных компонентов:

- сервера метаданных;
- серверов хранения объектов;
- клиентов *Lustre* [2].

*Integrated Manager for Lustre (IML)* — это набор инструментов управления и мониторинга, которые обеспечивают единое, единое представление *Lustre*. Он упрощает установку, настройку, мониторинг и общее управление *Lustre*. В нем предусмотрен мониторинг предупреждений и метрик в реальном времени с помощью богатой визуализации и программируемого API. На рисунке 1 представлен скриншот интерфейса

*General Parallel File System (GPFS)*.

*General Parallel File System (GPFS)* — это проприетарная высокопроизводительная файловая система, разработанная компанией *IBM*. Она широко используется в суперкомпьютерных средах, крупномасштабной

аналитике данных и облачных средах. *GPFS* разработана для обеспечения параллельного доступа к данным на нескольких узлах кластера [3].

*IBM Spectrum Scale GUI* — это веб—инструмент, который предоставляет администраторам графический интерфейс для управления файловыми системами *GPFS*. *Spectrum Scale GUI* позволяет администраторам отслеживать производительность системы, настраивать файловые системы, управлять пользователями и группами.

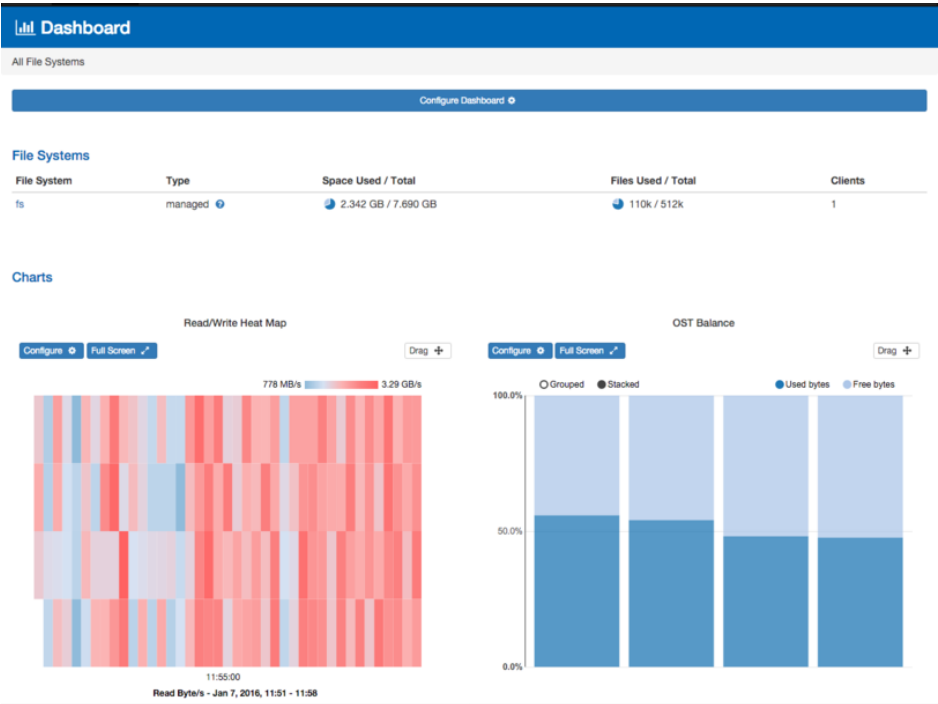


Рисунок 1 — Пользовательский интерфейс IML

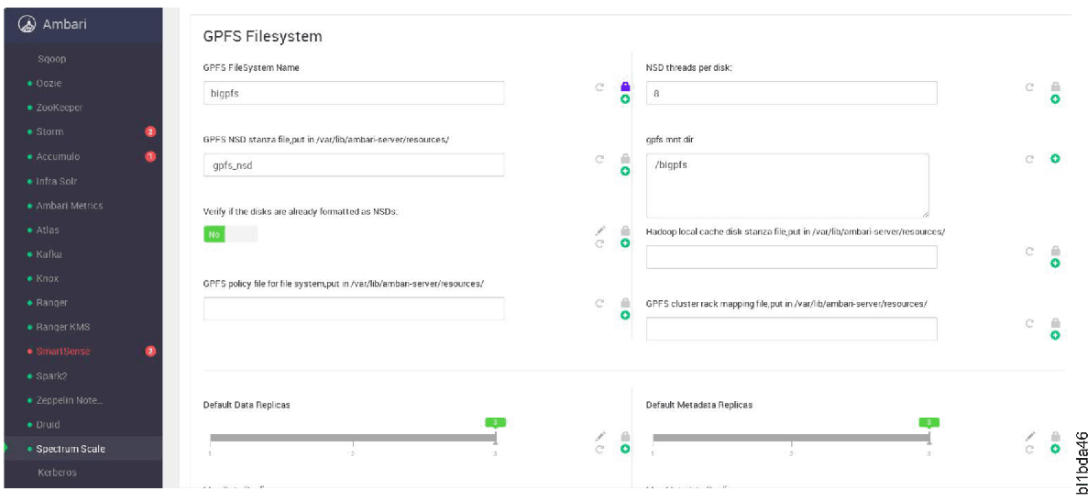


Рисунок 2 — *IBM Spectrum Scale GUI*

## *GlusterFS.*

*GlusterFS* — это распределенная файловая система с открытым исходным кодом, предназначенная для масштабирования до петабайтов данных. *GlusterFS* позволяет администраторам управлять хранилищем как единым пулом и обеспечивает высокую доступность и отказоустойчивость за счет избыточности [4].

*Gluster Management Console (GMC)* — это веб-интерфейс, который позволяет администраторам управлять и контролировать файловые системы *GlusterFS*. С помощью GMC можно контролировать производительность системы, просматривать системные журналы, управлять пользователями и группами.

Также *GlusterFS* имеет интерфейс командной строки *gluster*, который позволяет администраторам управлять и контролировать файловую систему. Команду *gluster* можно использовать для просмотра состояния системы, настройки файловой системы, управления пользователями и группами, а также для мониторинга производительности системы.

Синтаксис команд:

*gluster [commands] [options] [--remote-host=<gluster\_node>] [--mode=script] [-xml].*

Примеры команд:

- отобразить информацию обо всех томах или указанном томе  
*volume info [all|<VOLNAME>];*
- вывести список всех томов в кластере  
*volume list;*
- удалить указанный том  
*volume delete <VOLNAME>;*
- запустить указанный том  
*volume start <VOLNAME>;*
- остановить указанный том

*volume stop <VOLNAME> [force];*

- Создать новый том указанного типа

*volume create <NEW-VOLNAME> [stripe <COUNT>] [[replica <COUNT>  
[arbiter <COUNT>]] | [replica 2 thin-arbiter 1]];*

*Ceph.*

*Ceph* — это распределенная файловая система, состоящая из демона хранения объектов (*OSD*), сервера метаданных (*MDS*) и монитора (*MON*). *OSD* хранит данные, а *MDS* обрабатывает операции с метаданными. *MON* следит за состоянием кластера [5].

Утилитой управления этой системы является *Ceph Dashboard* — веб-интерфейс, который позволяет администраторам управлять файловыми системами *Ceph*, контролировать производительность системы, просматривать системные журналы и настраивать пользователей и группы.

*MooseFS.*

*MooseFS* — это отказоустойчивая, масштабируемая, сетевая файловая система. Она распределяет данные по нескольким физическим серверам, которые видны пользователю как один виртуальный диск. Совместима с *POSIX*, действует как любая другая *Unix*-подобная файловая система [6].

*MooseFS* имеет интерфейс командной строки *mfsmaster*, который позволяет контролировать файловую систему. В зависимости от параметров она может запустить, перезапустить или остановить главный процесс *MooseFS*.

Синтаксис команд:

*mfsmaster [-f] [-c CFGFILE] [-u] [-d] [-t LOCKTIMEOUT] [ACTION]*

*mfsmaster -s [-c CFGFILE]*

*mfsmaster -v*

*mfsmaster -h,*

где:

-v — вывести информацию о версии и выйти;

- h — вывести информацию об использовании и выйти;
- f — принудительно запустить главный процесс *MooseFS*, не пытаясь убить предыдущий экземпляр;
- s — остановить главный процесс *MooseFS*;
- u — регистрировать неопределенные значения конфигурации (когда принимаются значения по умолчанию)
- c <CFGFILE> — указать альтернативный путь к файлу конфигурации.

Исходя из вышеперечисленных примеров, видно, какой функционал востребован для рассматриваемого класса утилит, а именно:

- запуск, безопасное завершение работы системы, гибернация;
- создание, редактирование, удаление томов (или иных типов хранилищ);
- мониторинг нагрузки, доступности компонентов систем.

Среди типов пользовательского интерфейса утилит можно выделить следующие:

- WEB-клиент;
- Консольная утилита

Первый тип, в силу богатого веб-инструментария для создания GUI, больше подходит для мониторинга, однако некоторые управляющие функции всё же удобнее делать в графических приложениях, например, когда от пользователя требуется ввод большого количества параметров. Второй же больше подходит для управления системой. Последовательность консольных команд удобно агрегировать в скрипты, что позволяет мощно автоматизировать задачи. В некоторых системах, консольные и графические приложения дублируют функционал, что расширяет варианты использования.

В рамках целевой кластерной файловой системы — *CSFS*, предпочтительным является консольный интерфейс, т.к. система активно развивается и новый функционал быстрее добавлять в консольную утилиту.

### 1.1.2 Постановка задачи на разработку

*QR ОС* (от англ. *QR – quality protect* «качество защиты») – это защищенная операционная система, имеющая в своей основе оригинальные, заново разработанные алгоритмы и компоненты, включая ядро системы. *QR ОС* создана в России «с нуля» на научно-техническом предприятии «Криптософт», не основана на других операционных системах и является полностью отечественной операционной системой [7].

Полнофункциональный набор компонентов системы позволяет решать широкий круг системных и прикладных задач и использовать *QR ОС* в серверах, автоматизированных рабочих местах, а также в качестве встраиваемой системы аппаратно-программных комплексов.

*QR ОС* соответствует требованиям ФСБ России к программному обеспечению, используемому в информационных и телекоммуникационных системах специального назначения и может использоваться для обработки информации, содержащей сведения, составляющие государственную тайну.

*QR ОС* является современной универсальной многозадачной операционной системой общего назначения, работающей по принципу разделения времени.

*QR ОС* предназначена для управления аппаратурой компьютера, обеспечения универсального интерфейса между аппаратурой и прикладными программами, создания универсальной вычислительной среды для прикладных программ, организации интерфейса между программами и пользователями компьютера.

Операционная система *QR ОС* отвечает требованиям, предъявляемым к современным операционным системам общего назначения, а именно:

- надежность и отказоустойчивость;
- безопасность;
- производительность;
- совместимость;

- расширяемость и переносимость;
- соответствие специальным требованиям.

На рисунке 3 изображен скриншот рабочего стола операционной системы с запущенными на ней приложениями.

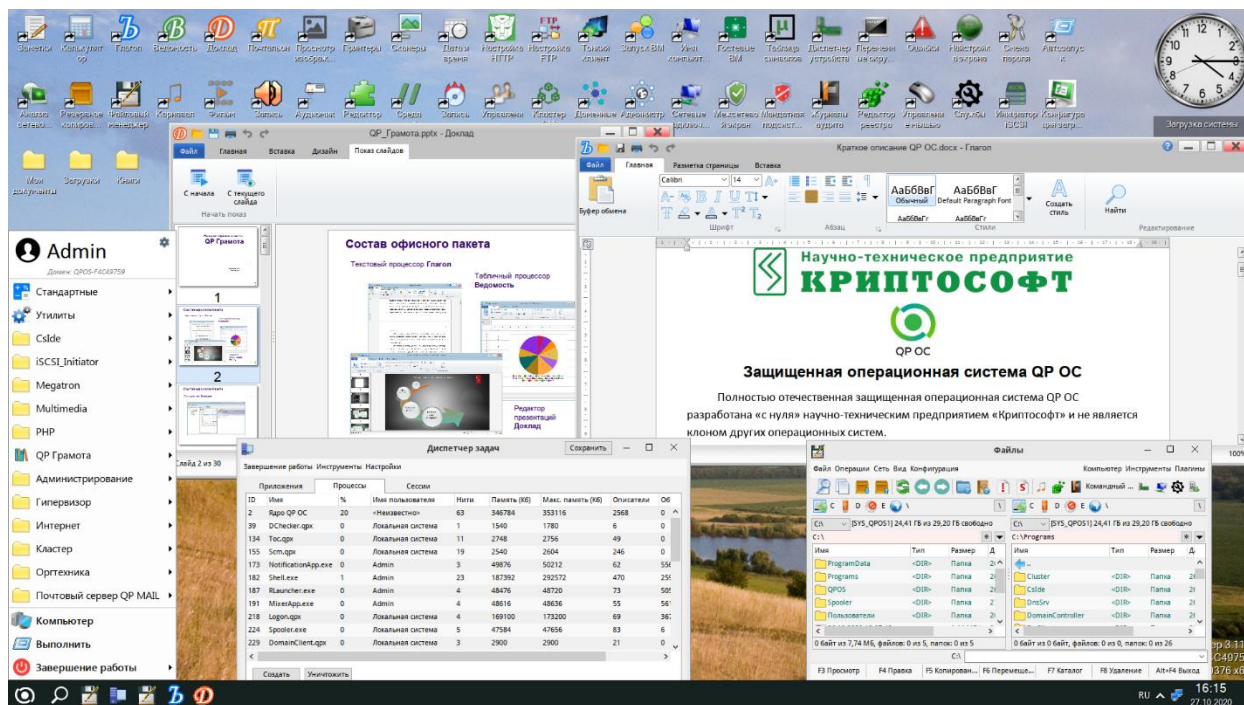


Рисунок 3 — Рабочий стол и запущенные приложения QR ОС

На данный момент для ОС QR ОС разрабатывается кластерная файловая под названием CSFS. Имеется возможность доступа к созданному хранилищу посредством протокола SMB путём трансляции SMB-пакетов в соответствующие запросы к клиенту файловой системы.

По аналогии с существующими кластерными файловыми системами, в CSFS выделяются компоненты четырёх типов, которые будут рассмотрены далее.

Мастер – основной узел кластера, который реализует хранение метаданных о файлах: дерева каталогов, адреса всех копий объектов файлов.

Клиент – узел кластера, который реализует подключение к мастеру кластера файловой системы и выполняет создание и монтирование виртуального диска, который является отображением файлов и каталогов, размещенных на ресурсах кластерной файловой системы.

Монитор – вспомогательный узел, который выполняет функцию мониторинга состояния группы размещения, перенаправления запросов узлам и агрегирования информации о них. Обменивается данными с мастером и подчиненными узлами.

Узел (хранилище) – узел кластера, который хранит данные файлов в виде объектов. Узел не хранит метаданные. В выделенном хранилище резервируется область для хранения битовой карты занятости объектов. В качестве узла хранения может быть использован физический или логический диск, файл-контейнер, *RAID*-массив, а также, например, сетевой диск, подключенные по протоколу *iSCSI* или *FibreChannel*.

Хранилище данных не содержит какой-либо файловой системы. Вместо этого используется специальная разметка носителя, которая определяет секции двух типов: секции метаданных в двух экземплярах, к которым относится и битовая карта выделенных и свободных объектов, и секция самих объектов. Объект – блок данных файла фиксированного размера. Логически – объект кластерной файловой системы аналогичен понятию кластера диска. В этом состоит одно из ключевых отличий от большинства существующих распределенных файловых систем, где объекты хранятся в виде файлов на локальной файловой системе. Данное решение влечёт за собой ряд преимуществ и недостатков. К основным преимуществам можно отнести высокую скорость доступа к содержимому объекта, поскольку обращение идет напрямую к диску по индексу объекта, а также отсутствие зависимости от наличия драйвера конкретной файловой системы. Последнее преимущество также является и недостатком, так как многие файловые системы предоставляют средства журналирования, восстановления после сбоя и другие средства безопасности. В данном случае данные функции должна реализовывать сама кластерная файловая система.

Группа узлов хранения, между которыми осуществляется репликация данных, образует группу размещения. Таким образом, группа размещения – группа узлов, которая хранит копии одних и тех же объектов.

Количество узлов в группе размещения соответствует фактору репликации.

Фактор репликации – количество хранимых копий объектов, каждая из которых хранится на отдельном узле внутри группы. В то время как минимальный фактор репликации – минимальное количество активных узлов, при которых группа размещения продолжает функционировать. Как только количество узлов становится меньше минимального фактора репликации, группа размещения блокируется.

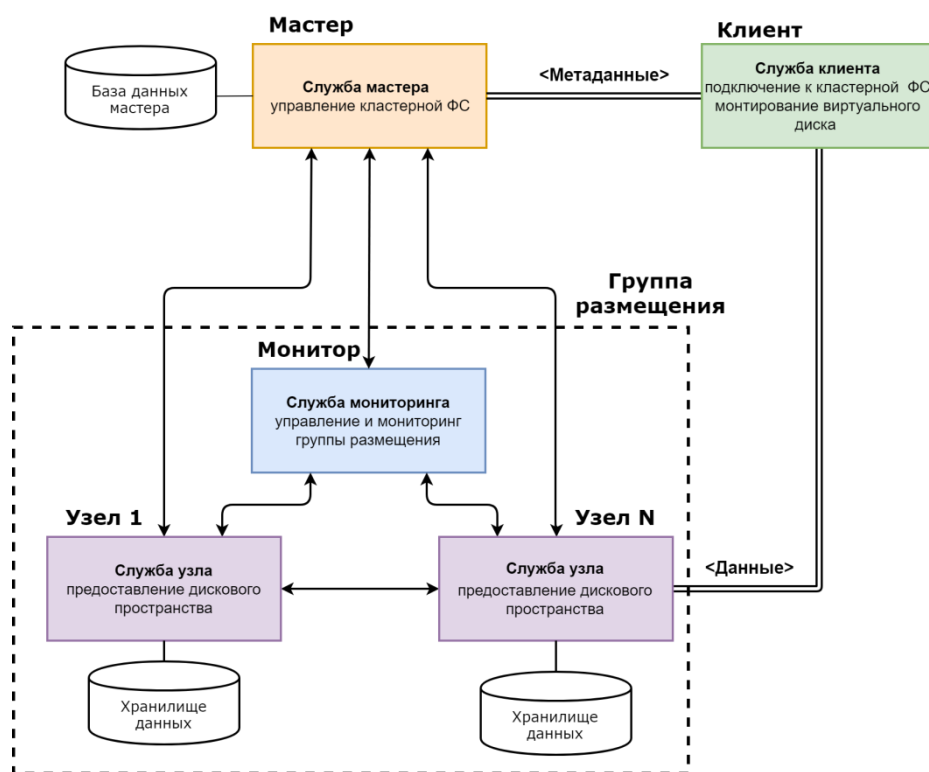


Рисунок 4 – Упрощенная схема CSFS

Чтобы определить место управляющей утилиты в контексте CSFS рассмотрим более подробную схему системы на рисунке 5.



В рамках данной работы, нас будет интересовать только взаимодействие службы мастера и утилиты управления, и косвенно служба Reserver. Все эти объекты и их взаимодействия представлены на рисунке 6.

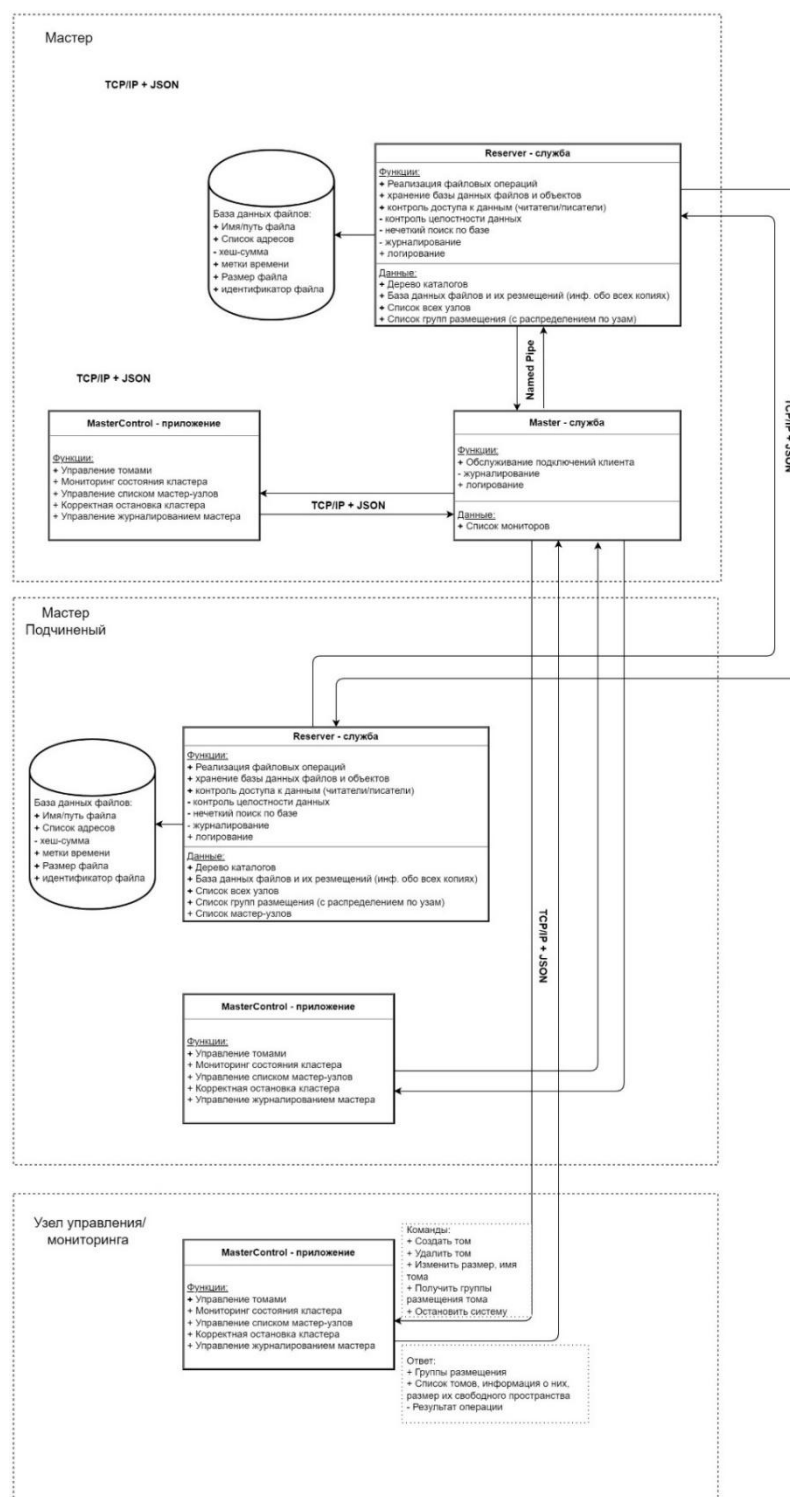


Рисунок 6 – Схема объектов CSFS, рассматриваемых в работе

*Reserver* – служба резервирования мастера. Реализует репликацию мастера, повышая тем самым, общую отказоустойчивость системы.

*MasterControl* – целевое приложение. Утилита управления кластерной системой *CSFS*.

Исходя из анализа существующих программных продуктов, и из особенностей системы, в которую необходимо интегрировать проект, сформулируем задачу.

Необходимо спроектировать и закодировать утилиту управления кластерной файловой системой *CSFS*.

Целевые пользователи программы: системные администраторы, сетевые администраторы, администраторы систем хранения данных.

Интерфейс программы: консольный, аргументы командной строки.

Требуемый функционал программы:

- создание и изменение томов с требуемой отказоустойчивостью;
- мониторинг томов;
- управление репликацией;
- мониторинг активных подключений узлов хранения, управления, клиентов;
- безопасное завершение работы *CFS*.

Команды могут исполняться как локально, так и отправляться удаленной службе. Взаимодействие должно происходить с помощью сокетов.

### 1.3.1 Выбор языка программирования

Исходя из поставленных задач, разрабатываемое приложение обладает следующими характеристиками:

- низкие требования к системным ресурсам. Предполагается, что утилита управления сама по себе не выполняет никаких ресурсоемких задач, а только лишь обеспечивает внешний доступ к системе по заданному протоколу;

— высокие требования к масштабируемости приложения. Основной объект программы – команда. Ввиду комплексности системы, которой утилита управляет, добавление новых команд и поддержка нового функционала *CFS* должны производиться быстро и без основательных архитектурных изменений;

— жесткая привязка к платформе. Хорошо, т.к. при разработке меньше необходимости уделять внимание кроссплатформенности. Плохо, т.к. сильно ограничен выбор инструментов.

Отдельно стоит отметить, что разработка *CSFS* изначально ведется на ОС *Windows*.

Для удовлетворения этих характеристик существует множество инструментов. Под инструментами имеем ввиду язык программирования и среду разработки для него.

*Python*.

Язык *Python* – высокоуровневый, практически скриптовой язык. Отличается высокой выразительностью своих конструкций, богатой стандартной библиотекой и наличием обширной базы сторонних библиотек. Хорошо подходит для решения данной задачи, т.к., например, модуль парсинга командной строки уже встроен в библиотеку языка.

Причины выбора *Python*:

– простота и легкость использования: *Python* имеет чистый и читаемый синтаксис, что делает его отличным выбором для новичков и быстрой разработки;

– обширная стандартная библиотека: *Python* поставляется с широким спектром встроенных модулей, которые обеспечивают функциональность для различных задач, уменьшая необходимость во внешних зависимостях;

– большое сообщество и экосистема: *Python* имеет живое и активное сообщество, что означает наличие большого количества ресурсов, библиотек и фреймворков для решения различных задач.

### Причины против выбора *Python*:

- производительность. *Python* — интерпретируемый язык, который может быть медленнее, чем компилируемые языки, такие как *Rust*, *Kotlin* или *C++*;
- глобальная блокировка интерпретатора (*GIL*). *GIL* в *Python* может ограничивать производительность многопоточности, что делает его менее подходящим для многопоточных приложений;
- потребление памяти. *Python* имеет тенденцию использовать больше памяти по сравнению с такими языками как *C++* или *Rust*.

### *IDE* для *Python*.

*PyCharm*, разработанный компанией *JetBrains*, популярная *IDE* для *Python*. Она предлагает ряд функций, специально предназначенных для разработки на *Python*, что делает ее мощным инструментом как для начинающих, так и для опытных разработчиков. Некоторые ключевые особенности включают:

- интеллектуальное завершение кода. *PyCharm* предоставляет точные и контекстно-зависимые предложения кода, что помогает повысить производительность и сократить количество ошибок при кодировании;
- надежная отладка. Встроенный отладчик позволяет просматривать код, устанавливать точки останова, проверять переменные и отслеживать выполнение программы для эффективной отладки;
- интегрированные инструменты тестирования. *PyCharm* поддерживает различные фреймворки тестирования, позволяя легко проводить модульное тестирование и разработку на основе тестирования (*TDD*) в *IDE*;
- интеграция контроля версий. *PyCharm* легко интегрируется с популярными системами контроля версий, такими как *Git*, что делает удобным управление репозиториями кода и совместную работу с членами команды.

*Rust.*

Язык *Rust* – популярное решение для создание системного ПО. Сочетает парадигмы функционального и процедурного программирования с объектной системой, основанной на типажах. Также является хорошим выбором для создания данного приложения. По сравнению с *Python* более низкоуровневый, однако может быть более удобен, при реализации сетевого взаимодействия с удаленной службой кластера по нестандартному прикладному протоколу.

Причины выбора *Rust*:

- производительность и безопасность памяти. *Rust* разработан для обеспечения как высокоуровневых абстракций, так и тонкого контроля над системными ресурсами, что приводит к созданию эффективного и безопасного кода;
- безопасность потоков. *Rust* обеспечивает соблюдение строгих правил владения и заимствования, предотвращая такие распространенные проблемы параллелизма, как гонки данных, что делает его хорошо подходящим для параллельных программ;
- сильное сообщество и инструментарий. Несмотря на то, что *Rust* является относительно новым языком, он обрел увлеченное сообщество, которое активно разрабатывает библиотеки и инструменты, обеспечивая прочную экосистему.

Причины против:

- кривая обучения. *Rust* имеет более сложную кривую обучения по сравнению с некоторыми другими языками из-за строгих правил владения и заимствования;
- время компиляции. Сильный акцент *Rust* на безопасность и производительность обходится ценой потенциально более длительного времени компиляции, особенно для больших проектов;

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 23   |

– меньшая экосистема библиотек: хотя экосистема *Rust* растет, она не может похвастаться такой же широтой и глубиной библиотек и фреймворков, как более известные языки, такие как *Python* или *C++*.

#### *IDE для Rust.*

Хотя *VS Code* и не является традиционной *IDE*, это очень популярный редактор кода с широкой поддержкой разработки на языке *Rust* через поддерживаемые сообществом расширения. Среди его особенностей можно выделить следующие:

- универсальность. *VS Code* - это легкий, но мощный редактор, поддерживающий множество языков программирования, что делает его гибким выбором для разработчиков;
- поддержка языка *Rust*. Сообщество *Rust* активно поддерживает расширение *Rust* для *VS Code*, предлагающее такие функции, как подсветка синтаксиса, завершение кода, проверка ошибок и навигация по коду;
- расширения и настройка. *VS Code* имеет обширную библиотеку расширений, позволяющую расширить его функциональность для разработки *Rust* или любого другого языка программирования.

#### *Kotlin.*

*Kotlin* - это современный статически типизированный язык программирования, разработанный компанией *JetBrains*. Он разработан для полной взаимозаменяемости с *Java*, что делает его удобным выбором для программирования общего назначения.

#### Причины выбора *Kotlin*:

- интероперабельность. *Kotlin* может быть легко интегрирован с существующими кодовыми базами *Java*, что делает его подходящим выбором, если вам необходимо взаимодействовать с библиотеками или фреймворками *Java*;

- лаконичность и удобочитаемость. *Kotlin* предлагает лаконичный и выразительный синтаксис, сокращая количество "котлового" кода и облегчая его понимание и сопровождение;

- современные возможности языка. *Kotlin* предоставляет такие современные возможности языка, как безопасность нуля, функции расширения и корутины, что повышает безопасность и выразительность кода.

Причины против выбора *Kotlin*:

- ограниченная экосистема. Хотя *Kotlin* набирает популярность, его экосистема может быть не такой обширной, как у таких языков, как *Python* или *C++*;

- производительность. *Kotlin*, как и *Java*, является языком, основанным на времени исполнения, поэтому в определенных сценариях он может не обеспечить такой же уровень производительности, как *Rust* или *C++*;

- потребление памяти. *Kotlin*, будучи языком, основанным на времени выполнения, может потреблять больше памяти по сравнению с языками более низкого уровня, такими как *Rust* или *C++*.

*IDE* для *Kotlin*.

*IntelliJ IDEA*, также разработанная компанией *JetBrains*, широко считается лучшей *IDE* для разработки *Kotlin*. Она предоставляет полный набор функций, повышающих производительность и поддерживающих специфические для *Kotlin* потребности разработки. Некоторые ключевые особенности включают:

- интеллектуальная помощь в кодировании. *IntelliJ IDEA* предлагает расширенные средства завершения кода, обнаружения ошибок и рефакторинга, которые повышают скорость разработки и помогают поддерживать качество кода;

- функции, специфичные для *Kotlin*. *IDE* обеспечивает отличную поддержку функций языка *Kotlin*, включая безопасность нуля, функции

расширения, корутины и многое другое, что позволяет эффективно использовать современные возможности *Kotlin*;

- бесшовная интеграция с Java. *IntelliJ IDEA* позволяет легко интегрироваться с существующими кодовыми базами Java, обеспечивая совместимость *Kotlin* и Java и облегчая работу над проектами на смешанных языках.

C++.

Язык C++ является универсальным языком программирования для создания обширного множества типов приложений, хотя более удобен для разработки системного ПО т.к. относится к типу компилируемых языков, код которого является неуправляемым.

Причины выбора C++:

- производительность. C++ – это компилируемый язык, который обеспечивает прямой доступ к аппаратному обеспечению, что делает его высокопроизводительным и подходящим для ресурсоемких приложений;

- низкоуровневый контроль. C++ позволяет осуществлять тонкий контроль над памятью и аппаратными ресурсами, что делает его подходящим для системного программирования или приложений, критичных к производительности.

Причины против выбора C++:

- сложность: C++ имеет сложный синтаксис и крутую кривую обучения, требующую тщательного управления памятью и большего внимания к деталям;

- отсутствие безопасности памяти: C++ не обеспечивает.

Язык отлично подойдет для решения поставленных задач, и именно он выбран для разработки проекта. Помимо перечисленных выше особенностей языка, выбор обусловлено также обширной кодовой базой на этом языке в решении, в которое будет интегрирован данный проект, и имеющимися

наработками в виде библиотек и аналогичных модулей (консольных утилит).  
Стандарт языка - C++17. Среда разработки - *Visual Studio* 2019.

*Visual Studio* – это комплексная *IDE* от *Microsoft*, широко используемая для разработки на C++ под *Windows*. Она предлагает ряд мощных инструментов, разработанных специально для разработки на C++. Некоторые особенности включают:

- богатый опыт редактирования кода. *Visual Studio* предоставляет интеллектуальные функции завершения кода, подсветки синтаксиса и навигации по коду, облегчая написание, понимание и сопровождение сложных кодовых баз C++;

- надежные возможности отладки. *IDE* предлагает продвинутый отладчик с такими функциями, как точки останова, окна наблюдения, анализ стека вызовов и проверка памяти, что позволяет эффективно отлаживать приложения на C++;

- интегрированные инструменты сборки. *Visual Studio* включает мощную систему сборки, которая выполняет сложные задачи компиляции, компоновки и развертывания, что делает сборку и развертывание C++ проектов удобным;

- профилирование и анализ производительности. *IDE* предоставляет инструменты для профилирования C++ кода с целью выявления узких мест в производительности, утечек памяти и других возможностей оптимизации;

- управление проектами и совместная работа: *Visual Studio* предлагает комплексные возможности управления проектами, включая интеграцию контроля версий, управление задачами и инструменты совместной работы.

## 1.4 Проектирование и разработка

### 1.2.1 Интерфейс

Интерфейс утилиты был определен как консольный. Соответственно, важным элементом программы является синтаксис аргументов командной строки. К требованиям, которым он должен удовлетворять отнесем следующее:

- читаемость. Лексемы команды и аргументов должны быть семантически однозначны и должны являться словарным словом английского языка, либо его сокращением;
- краткость лексем. Чем короче названия команд и аргументов, тем проще их запомнить;
- расширяемость. Добавление новых команд не должно приводить к внесению существенных изменений в синтаксис;
- преемственность. Синтаксис должен быть знаком пользователю из опыта работы с похожими утилитами, и просто работы с терминалом;
- соответствие стилю среды. В контексте администрирования распределенных вычислительных систем данная утилита может являться лишь одной из многих подобных утилит. Хорошо, если синтаксис ее аргументов будет похож на остальные программы способствуя формированию единой среды.

За основу будет взят синтаксис команд `bash` в `Linux`.

Собственно, синтаксис.

*longArgName* = --<слово>

*longArgName* = -<буква>

*command* = слово

*param* = [*longArgName* | *shortArgName*] [<значение>]

*paramsList* = *param* [*paramsList*]

*additionalParamsList* = *paramList*

*commandParamsList* = *paramList*

*syntax* = [*additionalParamsList*] (*command*) [*commandParamsList*]

*longArgName* - длинное имя аргумента – слово.

*shortArgName* - короткое имя аргумента – буква, обычно первая буква длинного аналога, но необязательно, т.к. в разных словах первые буквы могут быть одинаковы.

*command* - название команды

значение – числовое, строковое или ключевое слово (*true*, *false*, *on*, *off* и т.п.).

*paramsList* – список параметров

*additionalParamsList* – список дополнительных (глобальных) параметров

*commandParamsList* – список параметров команд.

Пример команды.

Создать том с именем “*Vol1*” с фактором репликации 2 и минимальным фактором репликации 1:

--ip “192.168.45.10” *addvol* “*Vol1*” -r 2 -m 2

Здесь [--ip “192.168.45.10”] – дополнительный параметр со значением эквивалентным адресу удаленной службы, *addvol* – имя команды, -r 2 -m 2 – параметры команды.

## 1.2.2 Общий алгоритм работы программы

На рисунке 7 представлен общий алгоритм работы программы.

### 1.2.1 Команды

Как было отмечено ранее, жизненный цикл программы, а, следовательно, и ее объектов сравнительно мал. Типичный вариант использования приложения – набор аргументов команды в терминале вручную, или передача их через файл, принятие программой этих аргументов, исполнение команды, вывод результата и/или статуса исполнения команды.

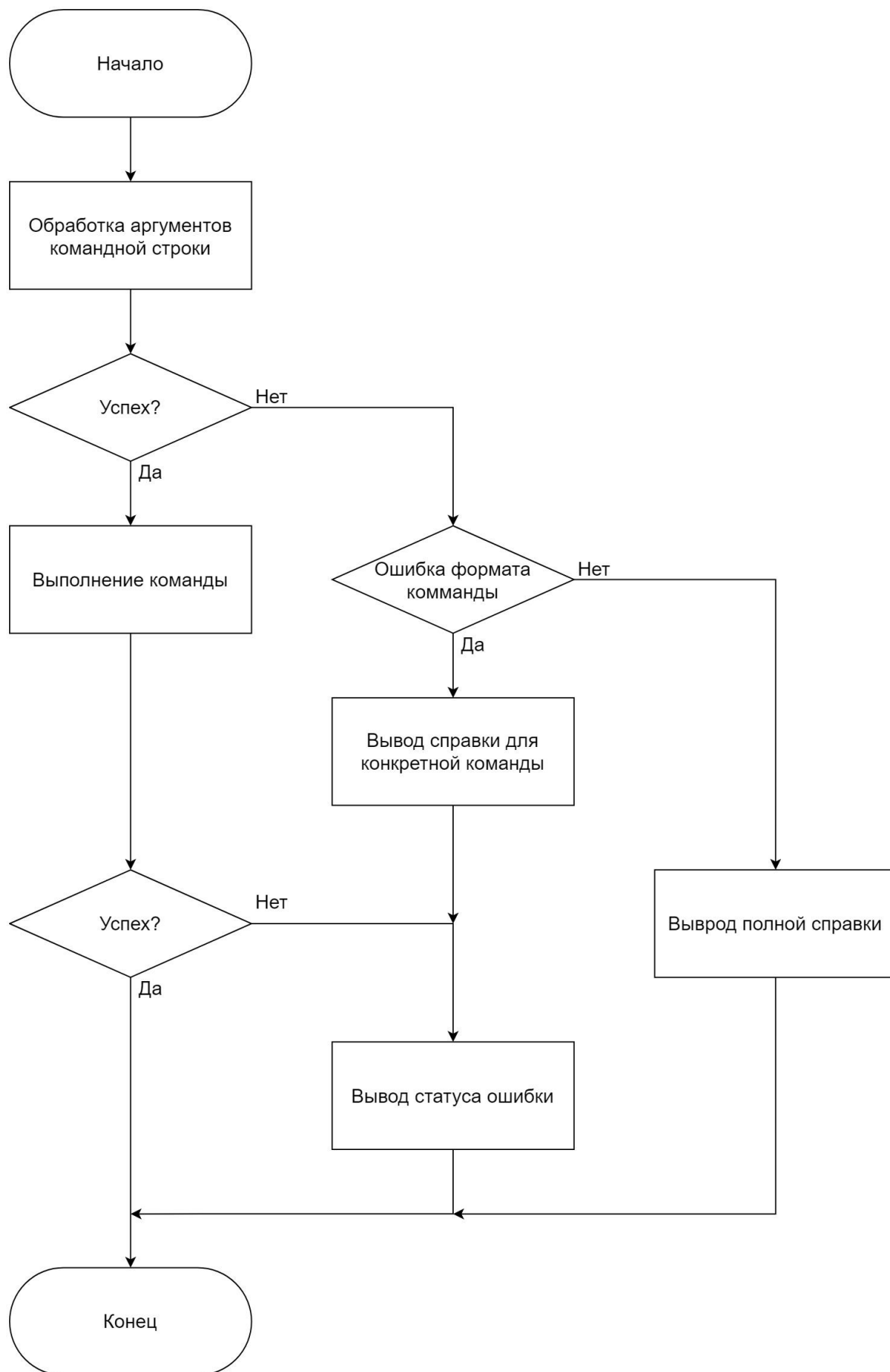


Рисунок 7 – Общий алгоритм работы программы

Каждая команда будет ассоциирована с классом, реализующим интерфейс, соответствующий заданному типу команды по исполнению. Типы команды по исполнению: *Local*, *Remote* – локальная и удаленная соответственно. Данные типы четко продиктованы заданными условиями. Очевидно, порядок их исполнения будет отличаться, но при этом, для всех команд одного этот порядок будет иметь много общего. Тогда будет уместно в соответствие с поставленной задачей, спроектировать иерархию классов команд, представленную на рисунке 8.

Команды типа *Remote* должны соответствовать конкретному типу запроса к Мастеру Csfs. Поэтому после парсинга они инициализируют данные с параметрами, если таковые имеются. Метод *GetRequestType* возвращает тип запроса. Перечисление с типами объявлено в протоколе взаимодействия с Мастером Csfs. Подробнее про каждую команду будет описано ниже.

### 1.2.2 Роль *CommandManager*.

Удобно добавлять функционал программы, в нашем случае команды, не внося изменения в существующий код, а лишь добавляя новые объект – в нашем единицы трансляции класса. Чтобы этого добиться, применим к объектам команд шаблон “Одиночка”, а доступ к этим объектам абстрагируем, через пул объектов, классом которого объявим статический класс *CommandManager*, у которого будут следующие методы:

```
void RegisterCommand(string commandName, Command& pCommand);
```

Зарегистрировать команду, а именно, добавить в таблицу, ключи в которой являются вводимые пользователем названия команд – *commandName*. Вызывается в конструкторе *AbstractCommand*. Таким образом, если класс конкретной команды действительно одиночка, тогда при инстанцировании, как глобальный объект, он регистрируется, при этом, нет необходимости явно добавлять код, связанный с этим.

```
Command& GetCommand(string commandName);
```

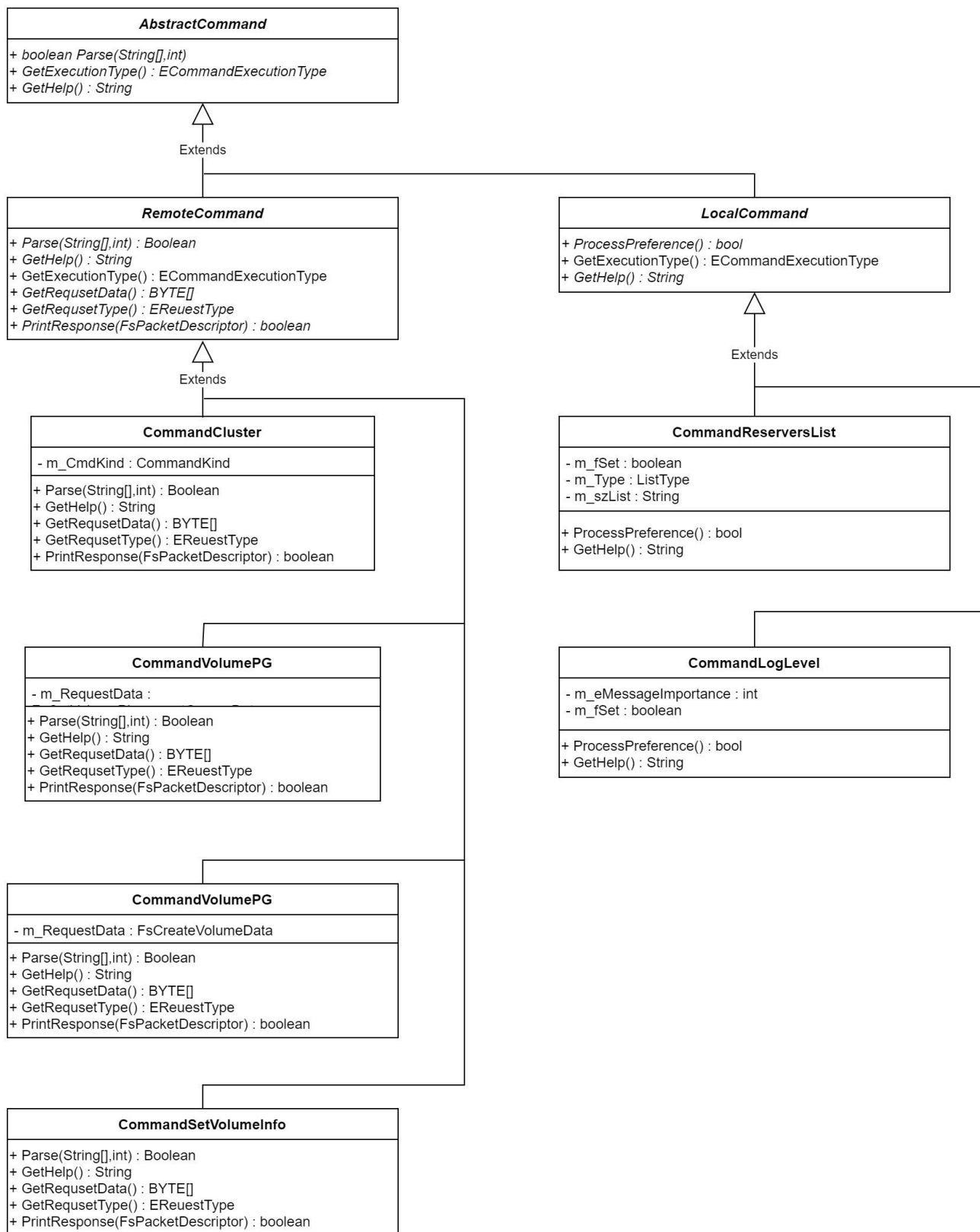


Рисунок 8 – Иерархия классов команд

Получить зарегистрированную команду по ключу *commandName*.  
Вызывается при парсинге команд.

```
List<Command&> GetAllCommands();
```

Получить все зарегистрированные команды. Удобно вызывать при выводе справочной информации о всех командах.

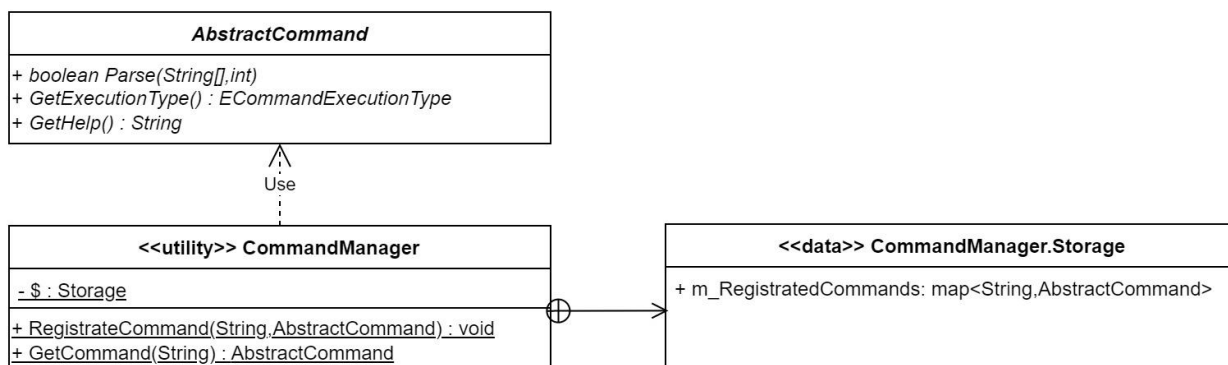


Рисунок 9 – Диаграмма менеджера команд

### 1.2.3 Парсинг аргументов командной строки

В C++ нет стандартных инструментов для обработки аргументов командной строки. В силу специфичности синтаксиса, разработаем свое решение данной задачи.

Исходя из определенного синтаксиса имеем алгоритм, схема которого представленно на рисунке 10. На рисунке 11 представленно диаграмма модуля парсинга.

### 1.2.4 Абстракция ключа

Ключ может быть коротким и длинным. Это существенно влияет на порядок обработки консольных аргументов, но для пользователя оба способа задания ключа равнозначны. Также, практически всегда короткий аналог ключа – первая буква длинного аналога, что было бы удобно эксплуатировать при объявлении. Необходимо учитывать также, что порядок ключей в списке не определен. Эти задачи решают вспомогательные классы *ParamKey* и *KeyExtractor*.

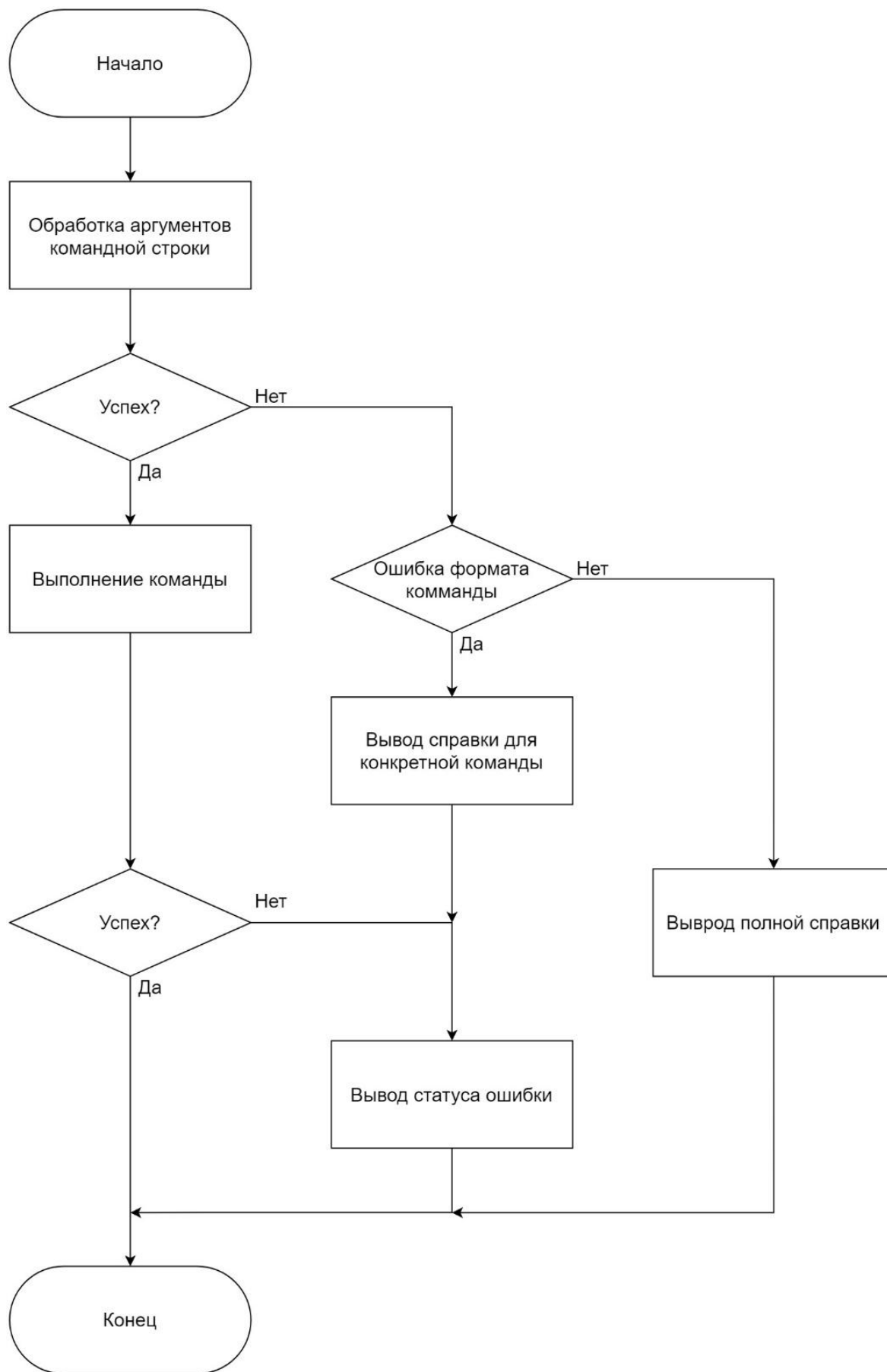


Рисунок 10 – Алгоритм парсинга аргументов командной строки

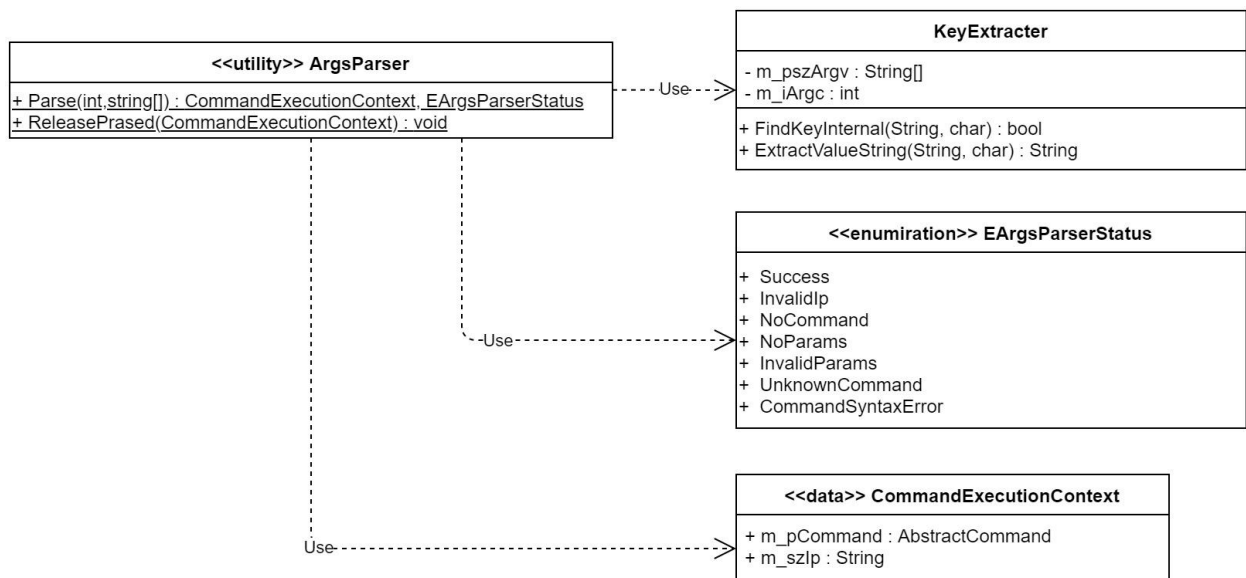


Рисунок 11 – Диаграмма модуля парсинга

Объявление шаблонной структуры *ParamKey*:

```

template<size_t LongKeyNameSize>
struct ParamKey {
public:
    constexpr ParamKey(const
wchar_t(&szLongKey)[LongKeyNameSize], wchar_t szShortKey) :
        cShortKey(szShortKey),
        szLongKey(ss(szLongKey))
    {}

    constexpr ParamKey(const
wchar_t(&szLongKey)[LongKeyNameSize]) :
        cShortKey(szLongKey[0]),
        szLongKey(ss(szLongKey))
    {}

public:
    wchar_t cShortKey;
    static_wstring_t<LongKeyNameSize> szLongKey;
};
  
```

Из определения видно, что данная структура - есть пара вида (символ, строка), при чем длинное имя ключа необходимо для объявления, и есть возможность явно задать короткое.

#### Объявление класса *KeyExtractor*:

```
class KeyExtractor {
public:
    KeyExtractor(int argc, const wchar_t* argv[]);
private:
    const wchar_t* ExtractValueStringInternal(
        const wchar_t* keyLong,
        const wchar_t keyShort
    ) const;

    bool FindKeyInternal(
        const wchar_t* keyLong,
        const wchar_t keyShort
    ) const;
public:
    template<size_t LongKeyLen>
    inline const wchar_t* ExtractValueString(const
ParamKey<LongKeyLen>& paramKey) {
        return
ExtractValueStringInternal(paramKey.szLongKey.data.data(),
paramKey.cShortKey);
    }

    template<size_t LongKeyLen>
    inline bool FindKey(const ParamKey<LongKeyLen>& paramKey) {
        return FindKeyInternal(paramKey.szLongKey.data.data(),
paramKey.cShortKey);
    }

private:
    int                m_iArgc;
```

```
const wchar_t** m_pszArgv;
};
```

Инстанцирование *KeyExtractor* происходит каждый раз, когда нужно обработать список ключей – т.е. в самом начале, обрабатывая дополнительный параметры, и при обработки конкретной команды. Как видно из объявления класса, публичные методы шаблонные и продолжают идею ParamKey – абстрагирование ключа.

Метод *ExtractValueString* возвращает строку, идущую в массиве *argv* за ключем – т.е значение параметра.

Метод *FindKey* возвращает истинность наличия заданного ключа в массиве *argv*.

### 1.2.5 Исполнение команды

Удаленные команды.

На рисунке 12 представлен алгоритма исполнения удаленной команды. На рисунке 13 изображена диаграмма модуля исполнения команд.

Подключения к службе происходит по ip-адресу полученному при обработки командной строки. Для исполнения удаленной команды необходимо сформировать пакет с запросом к службе мастера. Для этого необходим тип запроса, который получается из *RemoteCommand::GetRequestType()*, и данные, которые получаются из *RemoteCommand::GetRequestData()*. Данные могут отсутствовать.

*CommandCluster* – набор базовых команд кластера данных. Технически данный класс не соответствует конкретному запросу к мастеру, но каждый из его параметров – соответствует.

Список базовых команд:

- info - получить основную информацию о кластере;
- stop - безопасно остановить работу кластера полностью;
- pause - приостановить работу кластера.

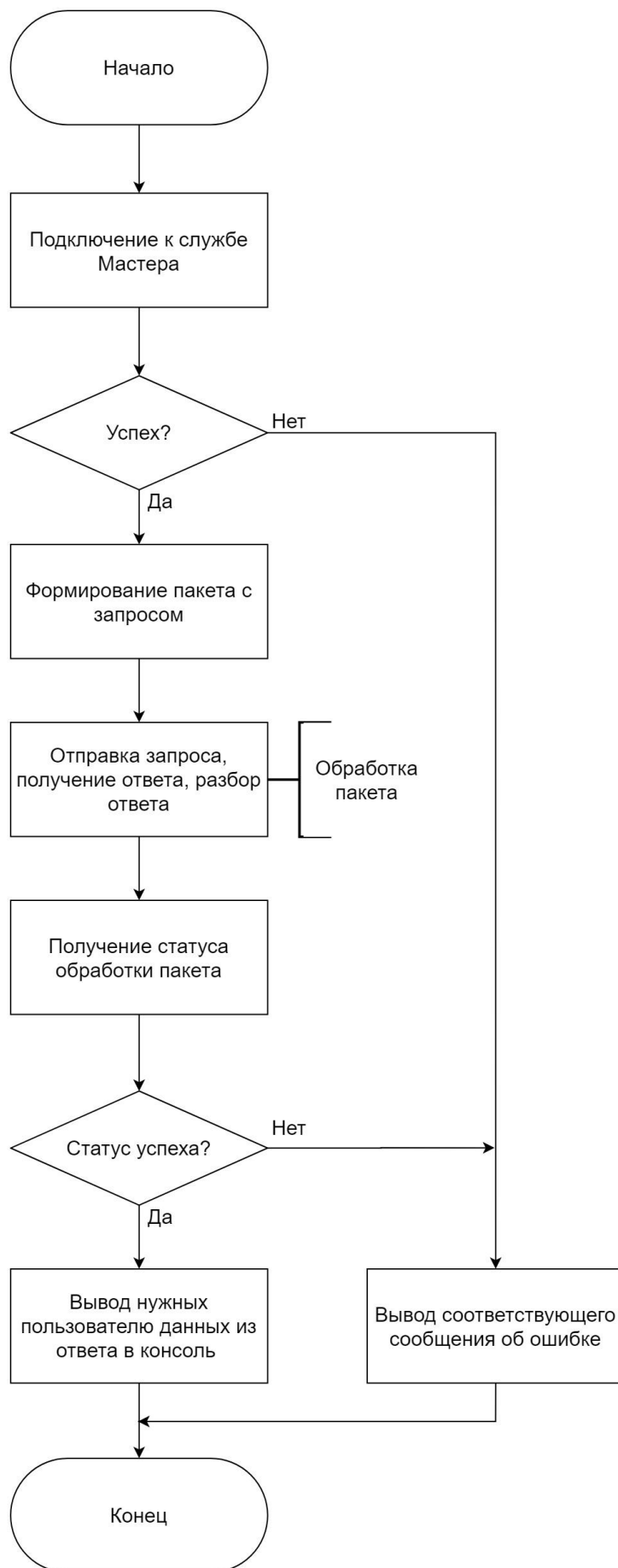


Рисунок 12 – Алгоритма исполнения удаленной команды

## Диаграмма модуля исполнения команд.

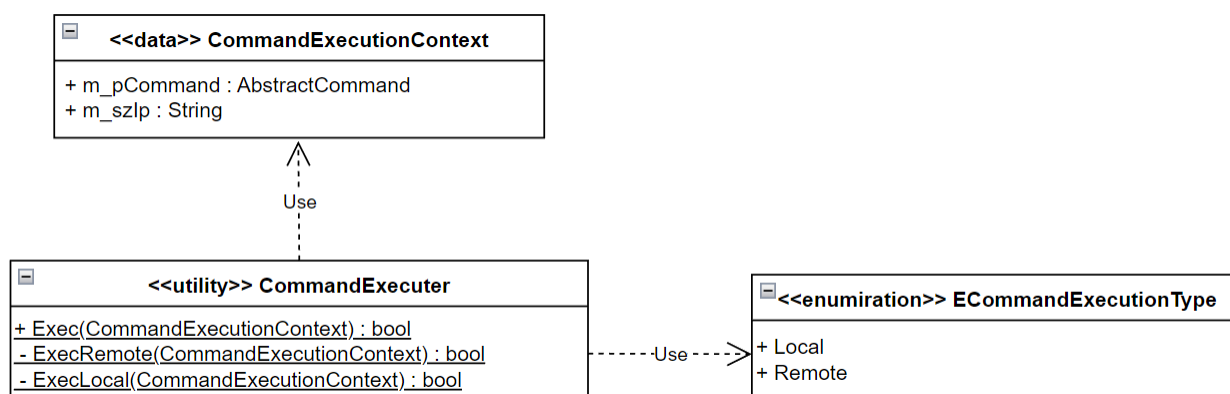


Рисунок 13 – Диаграмма модуля исполнения команд

*CommandCreateVolume* – команда создания тома CSFS.

*CommandSetVolumeInfo* – команда изменение параметров тома CSFS – имени, размера.

*CommandVolPg* – команда получения групп размещения тома CSFS.

Локальные команды.

Исполнение каждой локальной команды уникально, поэтому оно делегируется методу *LocalCommand::ProcessCommand()*.

И *CommanReserversList*, и *CommanLogLevel* работают с реестром, читая и изменяя соответствующие параметры.

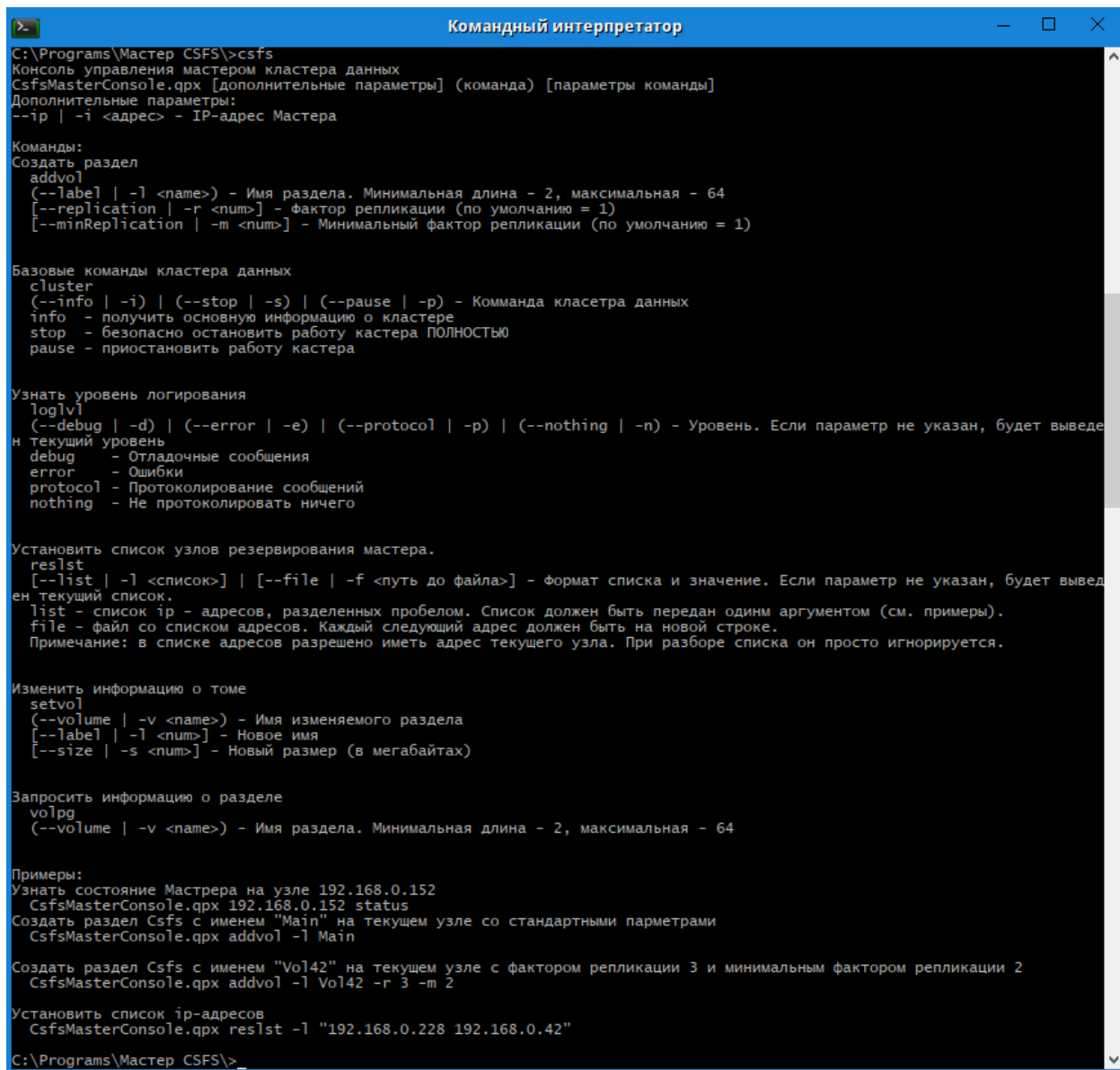
*CommanReserversList* – команда получение и изменения списка узлов реплики Мастера (ReserversNodes).

*CommanLogLevel* – команда получение и изменения уровня логирования Мастера CSFS.

### 1.3 Анализ результатов

Ниже представлены скриншоты терминала QP OS с работой программы.

На рисунке 14 выведена справка по всем командам.



```
C:\Programs\Мастер CSFS\>csfs
Консоль управления мастером кластера данных
CsfsMasterConsole.qpx [дополнительные параметры] (команда) [параметры команды]
Дополнительные параметры:
--ip | -i <адрес> - IP-адрес Мастера

Команды:
Создать раздел
addvol
  (--label | -l <name>) - Имя раздела. Минимальная длина - 2, максимальная - 64
  [--replication | -r <num>] - Фактор репликации (по умолчанию = 1)
  [--minReplication | -m <num>] - Минимальный фактор репликации (по умолчанию = 1)

Базовые команды кластера данных
cluster
  (--info | -i) | (--stop | -s) | (--pause | -p) - Команда кластера данных
info - получить основную информацию о кластере
stop - безопасно остановить работу кластера ПОЛНОСТЬЮ
pause - приостановить работу кластера

Узнать уровень логирования
loglvl
  (--debug | -d) | (--error | -e) | (--protocol | -p) | (--nothing | -n) - Уровень. Если параметр не указан, будет выведен текущий уровень
debug - Отладочные сообщения
error - Ошибки
protocol - Протоколирование сообщений
nothing - Не протоколировать ничего

Установить список узлов резервирования мастера.
reslstr
  [--list | -l <список>] | [--file | -f <путь до файла>] - Формат списка и значение. Если параметр не указан, будет выведен текущий список.
list - список ip - адресов, разделенных пробелом. Список должен быть передан одним аргументом (см. примеры).
file - файл со списком адресов. Каждый следующий адрес должен быть на новой строке.
Примечание: в списке адресов разрешено иметь адрес текущего узла. При разборе списка он просто игнорируется.

Изменить информацию о томе
setvol
  (--volume | -v <name>) - Имя изменяемого раздела
  [--label | -l <num>] - Новое имя
  [--size | -s <num>] - Новый размер (в мегабайтах)

Запросить информацию о разделе
volpg
  (--volume | -v <name>) - Имя раздела. Минимальная длина - 2, максимальная - 64

Примеры:
Узнать состояние Мастера на узле 192.168.0.152
CsfsMasterConsole.qpx 192.168.0.152 status
Создать раздел Csfs с именем "Main" на текущем узле со стандартными параметрами
CsfsMasterConsole.qpx addvol -l Main
Создать раздел Csfs с именем "Vol42" на текущем узле с фактором репликации 3 и минимальным фактором репликации 2
CsfsMasterConsole.qpx addvol -l Vol42 -r 3 -m 2
Установить список ip-адресов
CsfsMasterConsole.qpx reslstr -l "192.168.0.228 192.168.0.42"

C:\Programs\Мастер CSFS\>
```

Рисунок 14 – Справка по всем командам

На рисунке 15 выведена информации о кластере.

```

>
C:\Programs\Мастер CSFS>csfs cluster -i
Успешно

Незанятое пространство: 7.99  Гб

|Имя тома |Размер      |Свободно      |
|-----|-----|-----|
|Main     |7.99  Гб    |7.99  Гб      |

```

Рисунок 15 – Информации о кластере

Группа размещения тома

```

C:\Programs\Мастер CSFS>csfs volpg -v Main
Успешно

Фактор репликации:      1
Минимальный фактор репликации: 1

-----
Группа 1
-----
|LUN |Адрес      |Состояние      |Свободно      |Всего      |
|----|-----|-----|-----|-----|
|0   |3.1.150.140|              |7.99  Гб      |7.99  Гб    |

```

Рисунок 16 – Группа размещения тома

На рисунке 17 продемонстрировано добавление нового тома, и получение информации о нем.

```

>
C:\Programs\Мастер CSFS>csfs addvol -l Temp
Успешно

C:\Programs\Мастер CSFS>csfs cluster -i
Успешно

Незанятое пространство: 0.00  б

|Имя тома |Размер      |Свободно      |
|-----|-----|-----|
|Main     |7.99  Гб    |7.99  Гб      |
|Temp     |7.99  Гб    |7.99  Гб      |

```

```

C:\Programs\Мастер CSFS>csfs volpg -v Temp
Успешно

Фактор репликации:      1
Минимальный фактор репликации: 1

-----
Группа 1
-----
|LUN |Адрес      |Состояние      |Свободно      |Всего      |
|----|-----|-----|-----|-----|
|0   |3.1.150.142|              |7.99  Гб      |7.99  Гб    |

```

Рисунок 17 – Добавление нового тома и получили информацию о нем.

На рисунке 18 продемонстрированы уточнение синтаксиса конкретной команды и ее исполнение.

```
C:\Programs\Мастер CSFS\>csfs cluster
Неверный формат команды

Базовые команды кластера данных
info - получить основную информацию о кластере
stop - безопасно остановить работу кластера ПОЛНОСТЬЮ
pause - приостановить работу кластера
cluster
(--info | -i) | (--stop | -s) | (--pause | -p) - Команда кластера данных

C:\Programs\Мастер CSFS\>csfs cluster -s
Успешно
```

Рисунок 18 – Уточнение синтаксиса конкретной команды и исполнение ее

## 2 Охрана труда

Каждый день персональные компьютеры помогают людям решать различные задачи, будь то работа, учеба или развлечения. Однако, несмотря на все преимущества, использование компьютеров может нанести вред здоровью пользователя и окружающим людям. Чтобы минимизировать риск травм и заболеваний, необходимо строго соблюдать рекомендации по безопасности, такие как ограничение времени работы на компьютере, правильное расположение рабочего места, соблюдение норм освещения и уровня шума в помещении. Кроме того, важно следить за своим здоровьем и проводить регулярные перерывы для разминки и глазной гимнастики.

### 2.2 Основные вредоносные факторы

Правила выполнения работ с ПК обязаны обеспечивать минимизацию основных рисков, перечисленных в ГОСТ 12.0.003-2015. В первую очередь к ним относятся:

- повышенная температура отдельных элементов персонального компьютера;
- монотонность трудового процесса;
- высокий уровень зрительных нагрузок;
- вероятность поражения статическим электричеством, генерируемым техникой;
- недостаточная освещенность в зоне выполнения операций у компьютера;
- высокий уровень напряженности электрического и магнитного полей в зоне установки компьютера;
- другие факторы.

Полностью избавиться от вредного воздействия компьютера на здоровье невозможно, однако необходимо принимать меры для сокращения его воздействия на человека. Важно обеспечить возможность восстановления

|      |      |          |         |      |                              |      |
|------|------|----------|---------|------|------------------------------|------|
|      |      |          |         |      | <b>ПГУ1.090301.31.001 ПЗ</b> | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                              | 43   |

нормального самочувствия и работоспособности после работы за компьютером, не прибегая к дополнительным средствам, таким как массаж или медикаментозная терапия.

### **2.3 Общие требования к организации и оборудованию рабочих мест**

Действующее законодательство выделяет основные нормы, которых требует работа у компьютера в постоянном режиме:

- рабочие места изолированы друг от друга и обеспечиваются организованным воздухообменом. Если занимающие их сотрудники заняты творческой деятельностью, следует оборудовать их места перегородками высотой 1,5-2 метра;

- конструкция кресла регулируется, чтобы обеспечить соответствие его положения индивидуальным физическим параметрам сотрудника. Его поверхность должна быть нескользящей и гарантировать возможность легкой чистки;

- конструкция стола должна обеспечивать рациональное и удобное размещение имеющегося оборудования и иметь коэффициент отражения, не превышающий 0,7.

### **2.4 Требования к помещениям с компьютерами**

Для обеспечения безопасности при работе за компьютером устанавливаются определенные требования к помещениям, в которых происходит работа. Эти требования включают следующие меры:

- помещения должны иметь окна, через которые проходит естественный свет, которые должны быть ориентированы преимущественно на север или северо-запад. Если работа происходит в помещениях без естественного освещения, то необходимо провести высокоточные расчеты,

чтобы обосновать эквивалентность организованного искусственного освещения нормативам естественного света;

- окна в помещении должны быть оборудованы устройствами для регулирования яркости естественного освещения, такими как шторы, жалюзи и прочее;

- площадь одного рабочего места сотрудника должна составлять не менее 6 квадратных метров, а если в работе используются плоские мониторы компьютеров жидкокристаллического или плазменного типа, то не менее 4,5 квадратных метров;

- внутренняя отделка помещений должна производиться с использованием материалов, имеющих санитарно-эпидемиологические заключения. Максимальный коэффициент отражения используемых при отделке материалов составляет для потолка – 0,8, для стен – 0,6, для пола – 0,5;

- помещения должны находиться вдали от силовых кабелей, высоковольтных трансформаторов и другого оборудования, способного создавать помехи для офисной техники. В комнатах необходимо организовать защитное заземление, обеспечивающее безопасную работу компьютеров.

## 2.5 Правила размещения монитора

Согласно обязательным нормам техники безопасности, при размещении монитора необходимо соблюдать следующие требования:

- расстояние между столами составляет не менее 2 метров, а между боковыми поверхностями мониторов – не менее 1,2 метра;

- экран используемого монитора находится на расстоянии 0,6-0,7 метра от глаз.

При этом электростатический потенциал дисплея не должен превышать 500 В.

|      |      |          |         |      |                              |      |
|------|------|----------|---------|------|------------------------------|------|
|      |      |          |         |      | <b>ПГУ1.090301.31.001 ПЗ</b> | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                              | 45   |

## 2.6 Режим труда и отдыха при использовании компьютера

В инструкции по охране труда содержатся правила безопасного поведения и техники выполнения трудовых операций. В зависимости от категории выполняемой работы и уровня зрительной нагрузки, которая возникает у пользователя в течение смены, существует определенное время перерывов, которое зависит от продолжительности работы:

- при 8 - часовой смене – от 30 до 70 минут;
- при 12 - часовой смене – от 70 до 120 минут.

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 46   |

## Заключение

В ходе выполнения данной работы изучены способы организации распределенных вычислительных систем, получены знания по взаимодействию кластерными файловыми системами и проанализированы существующие утилиты управления ими.

Также получены навыки написания консольного приложения на языке программирования C++, изучены и применены методы написания проектов с помощью библиотеки *WinAPI*. Освоен функционал библиотеки для управления состояниями приложения *Winsock*.

Во время выполнения работы был разработана утилита управления кластерной файловой системой *CSFS*, которая позволяет пользователю управлять системой и выполнять мониторинг ее состояния.

В дальнейшем возможно беспрепятственное расширение функционала программы, в соответствие с усовершенствованием сопряженной с ней *CFS*.

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 47   |

## Список использованных источников

1. Маркелов А.А. OpenStack. Практическое знакомство с облачной операционной системой. - 4-е издание. ДМК Пресс. 2018. - 306 с.
2. Документация по *Lustre*. [Электронный ресурс] / Режим доступа: <https://www.lustre.org/> свободный. (дата обращения 06.03.2023).
3. Документация по *GPFS*. [Электронный ресурс] / Режим доступа: <https://www.ibm.com/docs/en/gpfs/4.1.0.4?topic=system-basic-gpfs-structure> свободный. (дата обращения 06.03.2023).
4. Документация по *GLuster*. [Электронный ресурс] / Режим доступа: <https://manpages.debian.org/unstable/glusterfs-cli/gluster.8.en.html> свободный. (дата обращения 06.03.2023).
5. Документация по *Ceph*. [Электронный ресурс] / Режим доступа: <https://ceph.io/> свободный. (дата обращения 06.03.2023).
6. Документация по *MooseFS*. [Электронный ресурс] / Режим доступа: <https://moosefs.com/Content/Downloads/moosefs-3-0-users-manual.pdf>. (дата обращения 06.03.2023).
7. Обзор операционной системы *QP ОС*. [Электронный ресурс] / Режим доступа: <https://cryptosoft.ru/qpos.html> свободный. (дата обращения 06.03.2023).
8. Bjarne Stroustrup, Herb Sutter, Gabriel Dos Reis. A brief introduction to C++'s model for type- and resource-safety. [Электронный ресурс] / Режим доступа: <https://www.stroustrup.com/resource-model.pdf>. (дата обращения 06.03.2023).

# Приложение А

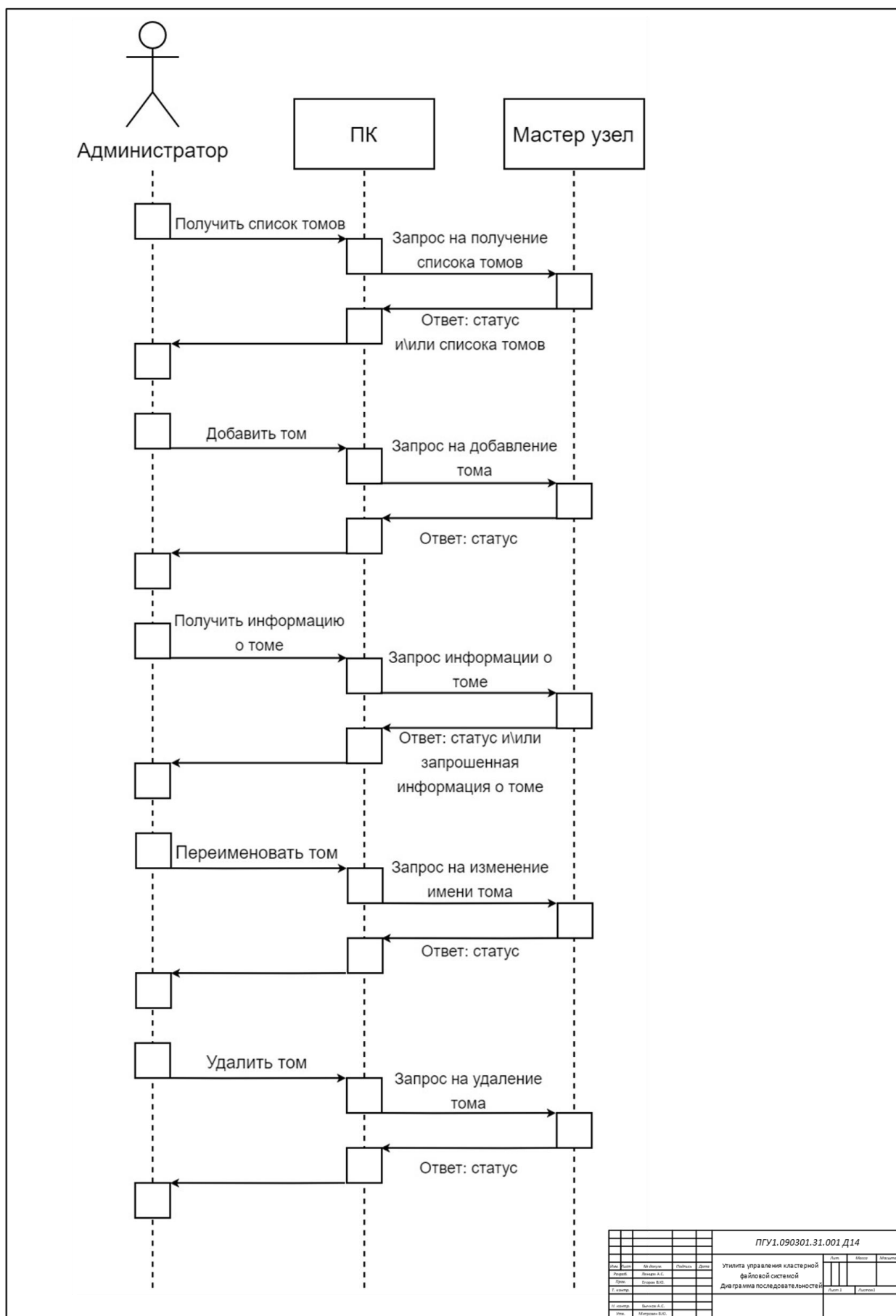
(Обязательное)

Утилита управления кластерной файловой системой

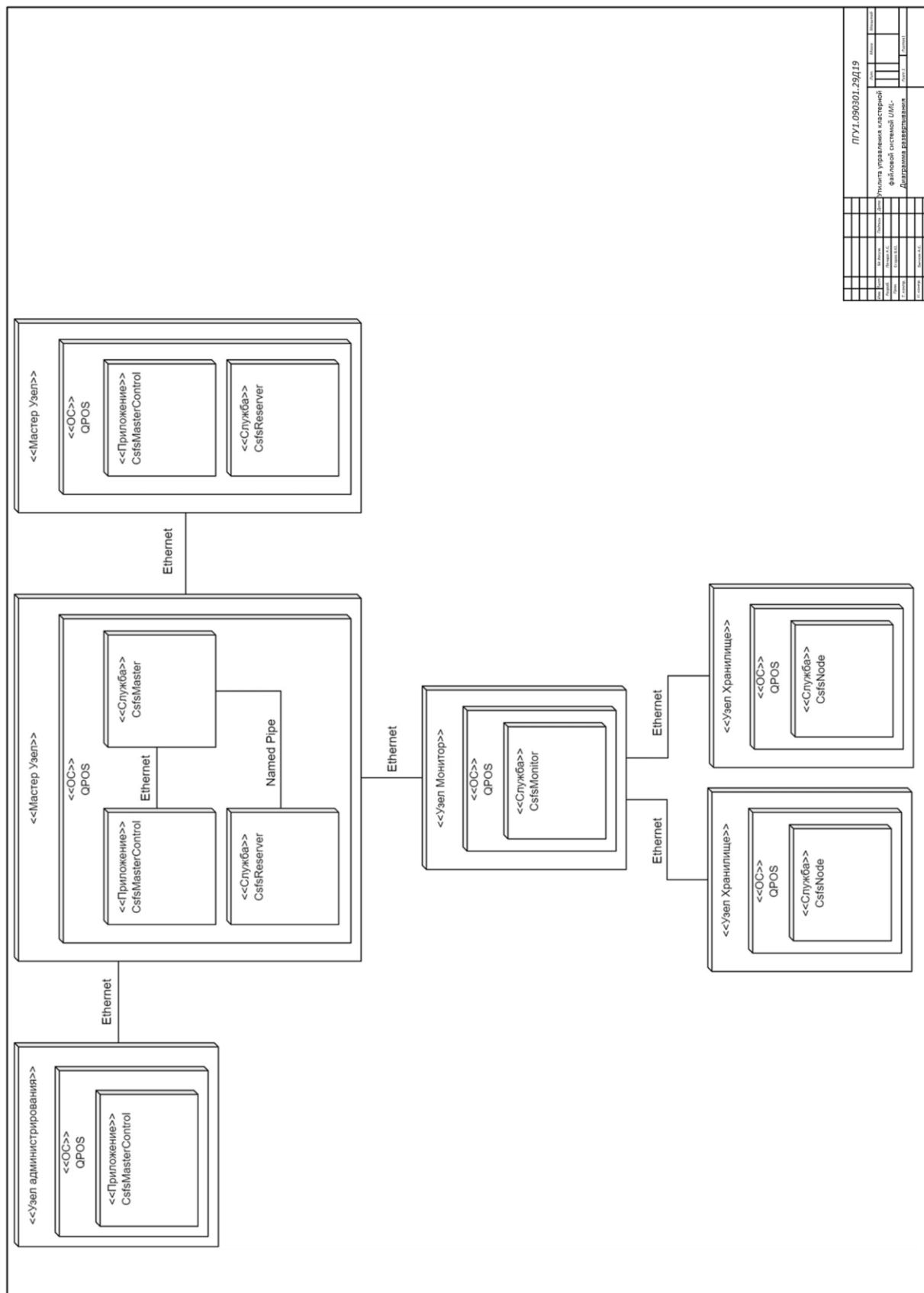
*UML*-Диаграммы

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
|      |      |          |         |      |                       | 49   |
| Изм. | Лист | № докум. | Подпись | Дата |                       |      |

## Приложение А.1 – Диаграмма последовательностей



## Приложение А.2 – Диаграмма развертывания

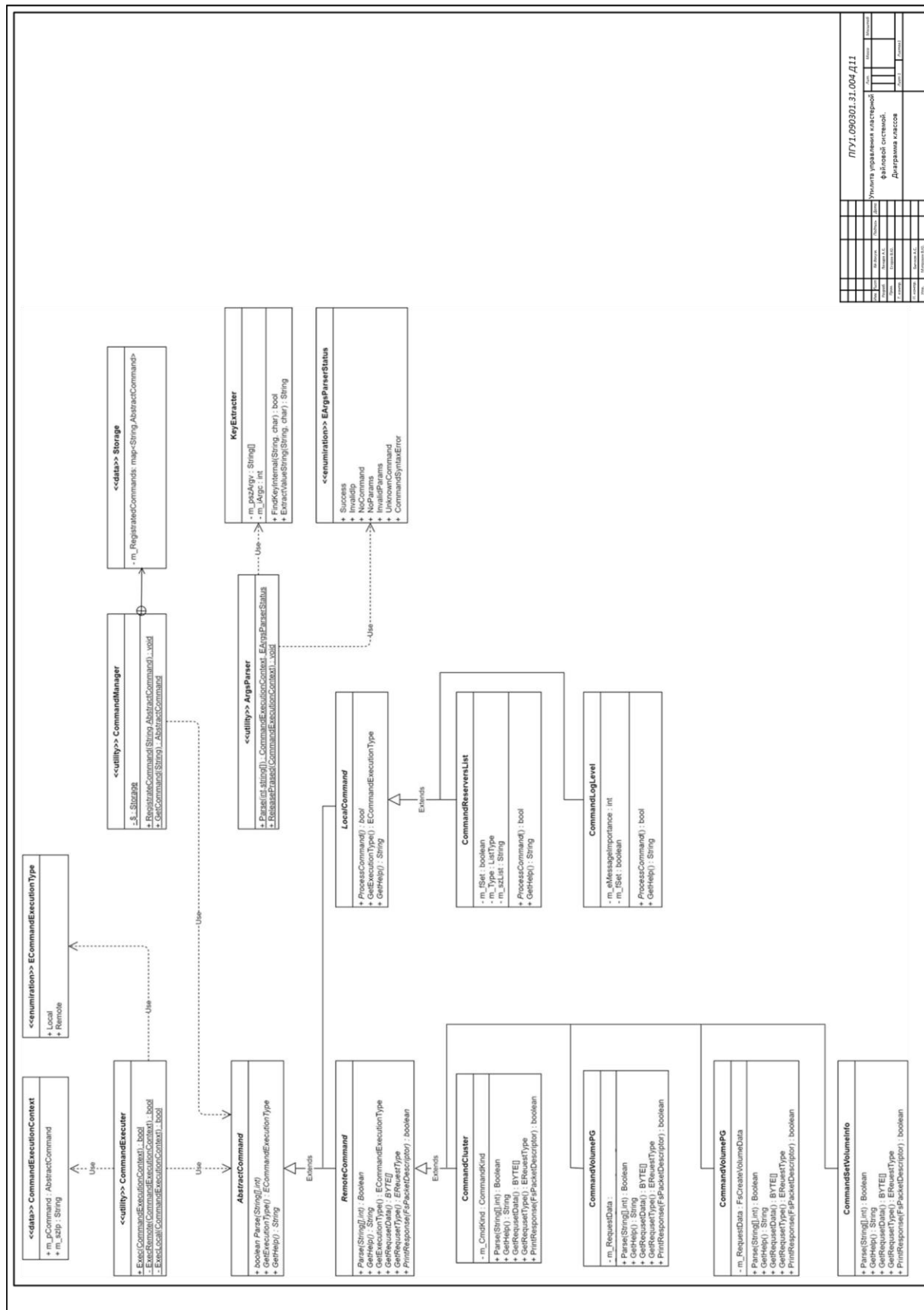


|                   |      |          |         |      |      |      |          |         |      |
|-------------------|------|----------|---------|------|------|------|----------|---------|------|
| ПУУ1.090301.29d19 |      |          |         |      |      |      |          |         |      |
| Изм.              | Лист | № докум. | Подпись | Дата | Изм. | Лист | № докум. | Подпись | Дата |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |
|                   |      |          |         |      |      |      |          |         |      |

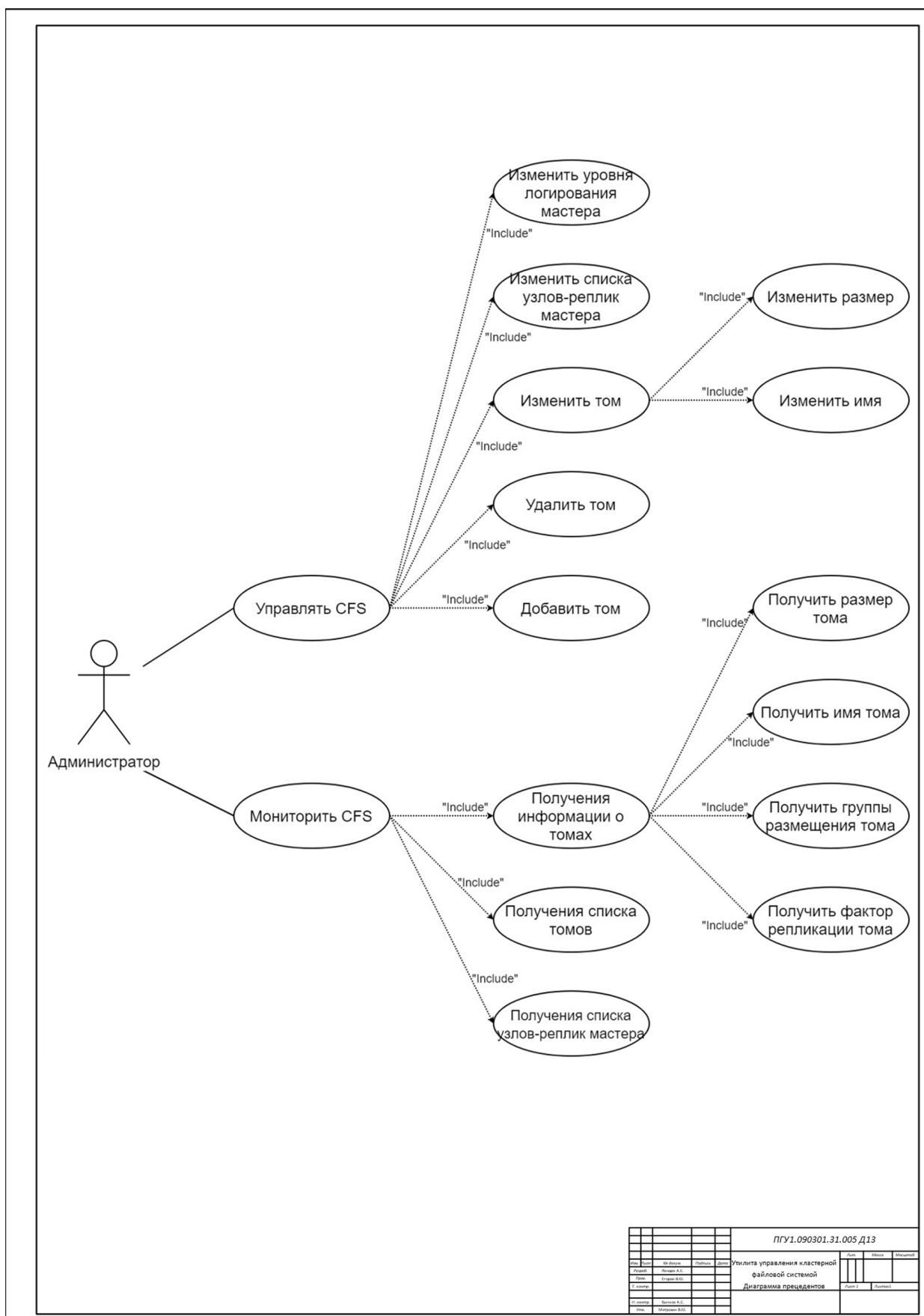
|      |      |          |         |      |
|------|------|----------|---------|------|
| Изм. | Лист | № докум. | Подпись | Дата |
|      |      |          |         |      |

ПУУ1.090301.31.001 ПЗ

## Приложение А.3 – Диаграмма классов

[illegible]

## Приложение А.4 – Диаграмма прецедентов



## Приложение Б

(Обязательное)

Утилита управления кластерной файловой системой

Листинг программы

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
|      |      |          |         |      |                       | 54   |
| Изм. | Лист | № докум. | Подпись | Дата |                       |      |

## Файл ArgsParser.cpp

```
#include "StdAfx.h"
#include "ArgsParser.h"

ArgsParser::ArgsParser() {}

ArgsParser::~ArgsParser() {}

const wchar_t* ArgsParser::GetStatusString(EArgsParserStatus
status) {
    static const RangedEnumStrings<EArgsParserStatus>
s_StatusStrings{
        {EArgsParserStatus::Success,          L"Успех"},
        {EArgsParserStatus::InvalidIp,
L"Недопустимый ip-адрес"},
        {EArgsParserStatus::NoCommand,          L""},
        {EArgsParserStatus::InvalidParams,      L"Неверные
параметры"},
        {EArgsParserStatus::CommandSyntaxError, L"Неверный
формат команды"},
        {EArgsParserStatus::UnknownCommand,
L"Несуществующая команда"},
    };
    return s_StatusStrings[status].c_str();
}

EArgsParserStatus ArgsParser::Parse(int argc, const wchar_t*
argv[], PSResult pResult){
    if (!argc)
        return EArgsParserStatus::NoParams;

    int prevArgsCount = argc;
    auto stat = ParseAdditionalKeys(&argc, argv, pResult);
    if (stat != EArgsParserStatus::Success)
        return stat;
}
```

```

        argv += (prevArgsCount - argc);
        return ParseCommand(&argc, argv, pResult);
    }

EArgsParserStatus ArgsParser::ParseAdditionalKeys(int* pArgc,
const wchar_t* argv[], PSResult pResult) {
    int argc = *pArgc;
    KeyExtractor keyExtractor(argc, argv);

    auto szIp = keyExtractor.ExtractValueString(IpKey);

    if (!szIp) {
        strcpy(pResult->m_szIp, CommonHelper::LOCAL_HOST_ADDR);
    }
    else {
        int argLen = wcsnlen(szIp, MAX_IP_ADDR_LEN);
        if (argLen == MAX_IP_ADDR_LEN)
            return EArgsParserStatus::InvalidIp;

        UnicodeToAnsi(szIp, pResult->m_szIp, argLen);
        argc -= 2;
    }
    *pArgc = argc;

    return EArgsParserStatus::Success;
}

EArgsParserStatus ArgsParser::ParseCommand(int* pArgc, const
wchar_t* argv[], PSResult pResult) {
    int argc = *pArgc;
    pResult->m_pCommand = nullptr;
    if (argc < MinArgsCount)
        return EArgsParserStatus::NoCommand;

    pResult->m_pCommand = CommandManager::GetCommand(*argv);

```

```

        if (pResult->m_pCommand == nullptr) {
            return EArgsParserStatus::UnknownCommand;
        }

        argv++;
        argc--;

        auto stat = pResult->m_pCommand->Parse(argv, argc) ?
            EArgsParserStatus::Success :
            EArgsParserStatus::CommandSyntaxError;

        *pArgc = argc;

        return stat;
    }

    void ArgsParser::ReleasePrased(PResult pResult) {

```

## Файл ArgsParser.h

```

#pragma once

enum class EArgsParserStatus {
    Success,
    InvalidIp,
    NoCommand,
    NoParams,
    InvalidParams,
    UnknownCommand,
    CommandSyntaxError,
    RangedEnumEnd,
};

class ArgsParser {

```

```

public:
    ArgsParser();
    ~ArgsParser();

public:
    //=====
    // KeyExtractor context
    //=====

    static constexpr ParamKey IpKey = L"ip";

    static constexpr int MinArgsCount = 0;

public:
    using SResult = CommandExecutionContext;
    using PResult = PCommandExecutionContext;

    /*
        * Отпарсить аргументы командной строки
        * @param[in] argc - Количество аргументов командной
строки без учета пути исполняемого файла (исходный argc-1)
        * @param[in] argv - Аргументы (исходный argv-1)
        * @return
        * Если Success | CommandSyntaxError, то m_szIp и
m_pCommand валидны.
        * Если UnknownCommand, m_szIp валидный
        * В остальных случаях поля pResult не определено
    */
    static EArgsParserStatus Parse(int argc, const wchar_t*
argv[], PResult pResult);

    static void ReleasePrased(PResult pResult);

    static const wchar_t* GetStatusString(EArgsParserStatus
eStatus);

```

```

private:
    static EArgsParserStatus ParseAdditionalKeys(int* pArgc,
const   wchar_t* argv[], PSResult pResult);

    static EArgsParserStatus ParseCommand(int* pArgc, const
wchar_t* argv[], PSResult pResult);

private:
    static constexpr auto LONG_ARG_PREFIX_LEN = ELEMENTS_IN("--
") - 1;
    static constexpr auto MAX_PARAM_NAME_LEN = 256 +
LONG_ARG_PREFIX_LEN;
    static constexpr auto MIN_PARAM_NAME_LEN = 2;

};

```

### Файл AbstractCommand.cpp

```

#include "StdAfx.h"
#include "CommonHelper.h"
#include "Command.h"

```

```

AbstractCommand::AbstractCommand(const wchar_t* szCommandName) {
    CommandManager::RegistrateCommand(szCommandName, this);
}

```

AbstractCommand.h

```

#pragma once

```

```

/*

```

Команда - синглтон, указатель на которую хранится в словаре, ключ - имя исполняемой команды ("addvol", "volpg").

Добавление (регистрация) команды происходит в момент инстанцирования синглтона, как глобального объекта, а именно

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 59   |

в конструкторе базового класса Command. Таким образом, чтобы добавить новую команду в утилиту, необходимо просто создать новый класс унаследованный от нужного base класса Remote\Local и объявить глобальный объект с этим классом.

```
*/
```

```
class AbstractCommand {
public:
    explicit AbstractCommand(const wchar_t* szCommandName);
    virtual ~AbstractCommand() = default;

public:
    virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) PURE;

    virtual ECommandExecutionType GetExecutionType() const PURE;

    virtual const wchar_t* GetHelp() const PURE;

};
```

CommandCluster.cpp

```
#include "StdAfx.h"
#include "CommandCluster.h"

CommandCluster CommandCluster::s_Instance;

CommandCluster::CommandCluster() :
    Base(COMMAND_NAME),
    m_CmdKind()
{ }
```

```

bool CommandCluster::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    if (keyExtractor.FindKey(InfoKey)) {
        m_CmdKind = CommandKind::Info;
        return true;
    }

    if (keyExtractor.FindKey(StopKey)) {
        m_CmdKind = CommandKind::Stop;
        return true;
    }

    if (keyExtractor.FindKey(PauseKey)) {
        m_CmdKind = CommandKind::Pause;
        return true;
    }

    return false;
}

const wchar_t* CommandCluster::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CommandCluster>(
    CommandParam(
        L"Команда кластера данных\n"
        L"  info  - получить основную информацию о
кластере\n"
        L"  stop  - безопасно остановить работу кластера
ПОЛНОСТЬЮ\n"
        L"  pause - приостановить работу кластера",
        CommandParamMondatoryElem(InfoKey),
        CommandParamMondatoryElem(StopKey),
        CommandParamMondatoryElem(PauseKey)

```

```

        )

    );
    return helpBuf.data.data();
}

int GetMaxVolumeLabelLen(PSCsfsClusterState pClusterState) {
    int iMax = 0;
    for (auto& it : pClusterState->m_aVolumesDescriptor) {
        int len =
wcsnlen(it.m_szVolumeLabel,ELEMENTS_IN(it.m_szVolumeLabel));
        if (len > iMax)
            iMax = len;
    }
    return iMax;
}

bool CommandCluster::PrintResponse(PSCsfsPacketDescriptor
pResponse) {
    switch (m_CmdKind){
    case CommandKind::Info:
        return PrintResponseCommandInfo(pResponse);

    case CommandKind::Stop:
        break;

    case CommandKind::Pause:
        break;
    }

    return true;
}

bool
CommandCluster::PrintResponseCommandInfo(PSCsfsPacketDescriptor
pResponse) {

```

```

        auto pClusterStateData = (PSCsfsClusterState)pResponse-
>m_pResponseData;

        wchar_t
szPrintableSize[CommonHelper::MAX_PRINTABLE_SIZE_LEN] = { 0 };
        CommonHelper::GetPrintableSize(pClusterStateData-
>m_UndefinedBlocksCount * CsfsObjectSize, szPrintableSize);
        printf("Незанятое пространство: %ws\n\n", szPrintableSize);

        wchar_t szTableHeaderVolumeLabel[] = L"Имя тома";
        int iMaxLabelLen = GetMaxVolumeLabelLen(pClusterStateData) +
2;
        iMaxLabelLen =
max((int)ELEMENTS_IN(szTableHeaderVolumeLabel), iMaxLabelLen);

        wprintf(L"|%-*s|%-15s|%-18s|\n", iMaxLabelLen,
szTableHeaderVolumeLabel, L"Размер", L"Свободно");
        wprintf(L"-----
---\n");

        wchar_t* szPrintableTotalSize = szPrintableSize;
        wchar_t
szPrintableFreeSize[CommonHelper::MAX_PRINTABLE_SIZE_LEN] = { 0
};

        for (auto& it : pClusterStateData->m_aVolumesDescriptor) {
            if (it.m_szVolumeLabel[0] == 0)
                break;

            CommonHelper::GetPrintableSize((QWORD)it.m_TotalBlocksCount
* CsfsObjectSize, szPrintableTotalSize);

            CommonHelper::GetPrintableSize((QWORD)it.m_FreeBlocksCount *
CsfsObjectSize, szPrintableFreeSize);

```

```

        wprintf(L"|%-*s|%-15ws|%-18ws|\n",
                iMaxLabelLen,
                it.m_szVolumeLabel,
                szPrintableTotalSize,
                szPrintableFreeSize
        );
    }

    return true;
}

ECsfsCommand CommandCluster::GetRequsetType() {
    return ECsfsCommand_ClusterState;
}

PBYTE CommandCluster::GetRequsetData() {
    return nullptr;
}

```

### Файл CommandCluster.h

```

#pragma once
#include "RemoteCommand.h"

class CommandCluster : public RemoteCommand {
public:
    using Base = RemoteCommand;

    CommandCluster();
    ~CommandCluster() = default;

public:
    static constexpr wchar_t COMMAND_NAME[] =
L"cluster";
    static constexpr wchar_t COMMAND_DESCRIPTION[] =
L"Базовые команды кластера данных";

```

```

enum class CommandKind {
    Info,
    Stop,
    Pause
};

static constexpr ParamKey InfoKey = L"info";
static constexpr ParamKey StopKey = L"stop";
static constexpr ParamKey PauseKey = L"pause";

public:
    virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) final;

    virtual const wchar_t* GetHelp() const final;

    virtual PBYTE GetRequasetData() final;

    virtual ECsfsCommand GetRequasetType() final;

    virtual bool PrintResponse(PSCsfsPacketDescriptor pResponse)
final;

private:
    bool PrintResponseCommandInfo(PSCsfsPacketDescriptor
pResponse);

private:
    static CommandCluster s_Instance;

private:
    CommandKind m_CmdKind;

};

```

```

CommandCreateVolume.cpp
#include "StdAfx.h"
#include "CommonHelper.h"
#include "CommandCreateVolume.h"

const CommandCreateVolume CommandCreateVolume::s_Instance;

CommandCreateVolume::CommandCreateVolume() :
    Base(COMMAND_NAME),
    m_RequestData()
{}

bool CommandCreateVolume::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    auto szValue =
keyExtractor.ExtractValueString(VolumeNameLongKey);
    if (szValue == nullptr) {
        return false;
    }

    int iLabelLen = wcslen(szValue);
    if (iLabelLen < 2)
        return false;

    wcsncpy(m_RequestData.m_wzsVolumeLabel, szValue, iLabelLen);
    m_RequestData.m_wzsVolumeLabel[CsfMaxVolumeName - 1] = 0;
    m_RequestData.m_nReplicationFactor = 1;
    m_RequestData.m_nMinimalReplicationFactor = 1;

    szValue =
keyExtractor.ExtractValueString(ReplicationFactorLongKey);

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 66   |

```

        if (nullptr != szValue) {
            if (!CommonHelper::_wtoi_s(
                szValue,
                CommonHelper::MAX_NUMBER_STRING_LEN,
                (int*)(&m_RequestData.m_nReplicationFactor)
            ))
                return false;
        }

        szValue =
keyExtractor.ExtractValueString(MinimalReplicationFactorLongKey)
;

        if (nullptr != szValue) {
            if (!CommonHelper::_wtoi_s(
                szValue,
                CommonHelper::MAX_NUMBER_STRING_LEN,
                (int*)(&m_RequestData.m_nMinimalReplicationFactor)
            ))
                return false;

            if (m_RequestData.m_nMinimalReplicationFactor >
                m_RequestData.m_nReplicationFactor) {

                m_RequestData.m_nMinimalReplicationFactor =
                    m_RequestData.m_nReplicationFactor;
            }

        }

        return true;
    }

    const wchar_t* CommandCreateVolume::GetHelp() const {
        static const auto helpBuf =
BuildCommandHelp<CommandCreateVolume>(

```

```

        CommandParam(
            L"Имя раздела. Минимальная длина - 2, максимальная
- 64",
            CommandParamMandatoryValue(VolumeNameLongKey,
L"name")
        ),
        CommandParam(
            L"Фактор репликации (по умолчанию = 1)",
            CommandParamOptionalValue(ReplicationFactorLongKey, L"num")
        ),
        CommandParam(
            L"Минимальный фактор репликации (по умолчанию =
1)",
            CommandParamOptionalValue(MinimalReplicationFactorLongKey,
L"num")
        )
    );
    return helpBuf.data.data();
}

bool CommandCreateVolume::PrintResponse(PSCsfsPacketDescriptor
pResponse) {
    return true;
}

ECsfsCommand CommandCreateVolume::GetRequestType() {
    return ECsfsCommand_CreateVolume;
}

PBYTE CommandCreateVolume::GetRequestData() {
    return reinterpret_cast<PBYTE>(&m_RequestData);
}

```

CommandCreateVolume.h

```
#pragma once
```

```
#include "RemoteCommand.h"
```

```
class CommandCreateVolume : public RemoteCommand {
```

```
public:
```

```
    using Base = RemoteCommand;
```

```
    CommandCreateVolume();
```

```
    ~CommandCreateVolume() = default;
```

```
public:
```

```
    static constexpr int MinArgsCount = 2;
```

```
    static constexpr wchar_t COMMAND_NAME[] =  
L"addvol";
```

```
    static constexpr wchar_t COMMAND_DESCRIPTION[] =  
L"Создать раздел";
```

```
    static constexpr ParamKey VolumeNameLongKey  
= L"label";
```

```
    static constexpr ParamKey ReplicationFactorLongKey  
= L"replication";
```

```
    static constexpr ParamKey MinimalReplicationFactorLongKey  
= L"minReplication";
```

```
public:
```

```
    virtual bool Parse(const wchar_t* pszParams[], int  
iParamsCount) final;
```

```
    virtual const wchar_t* GetHelp() const final;
```

```
    virtual PBYTE GetRequasetData() final;
```

```

        virtual ECsfsCommand GetRequsetType() final;

        virtual bool PrintResponse(PSCsfsPacketDescriptor pResponse)
final;

private:
        static const CommandCreateVolume s_Instance;

private:
        SCsfsCreateVolumeRequest m_RequestData;

};

```

### Файл CommandExecuter.cpp

```

#include "StdAfx.h"
#include "CommandExecuter.h"

bool CommandExecuter::Exec(PCommandExecutionContext
pParseResult){
    switch (pParseResult->m_pCommand->GetExecutionType()){
        case ECommandExecutionType::Remote:
            return ExecMasterRequest(pParseResult);

        case ECommandExecutionType::Local:
            return ExecPreferences(pParseResult);

        default:
            return false;
    }
}

bool CommandExecuter::ExecMasterRequest(PCommandExecutionContext
pCtx){
    char* szIp = pCtx->m_szIp;
    auto pCommand = static_cast<RemoteCommand*>(pCtx->
m_pCommand);

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 70   |

```

/* Not implemented */
if (pCommand->GetRequsetType() == ECsfsCommand_Invalid) {
    return true;
}

CsfsSynchroConnection pMasterConnection(szIp,
CsfsMakePort(CsfsMasterPort, CsfsMasterPort));
auto netStatus = pMasterConnection.Connect();
if (netStatus != ECsfsSynchroConnectionStatus::Success) {
    ConsoleControl::PrintNetStatus(netStatus);
    return false;
}

auto pRequestResponse = PacketBuilder::CreateRequest(
    pCommand->GetRequsetType(),
    pCommand->GetRequsetData(),
    &pMasterConnection
);

netStatus =
pMasterConnection.ProcessPocket(pRequestResponse);
ConsoleControl::PrintNetStatus(netStatus);
if (netStatus != ECsfsSynchroConnectionStatus::Success) {
    return false;
}

pCommand->PrintResponse(pRequestResponse);

PacketBuilder::DisposePacket(pRequestResponse);
return false;
}

bool CommandExecuter::ExecPreferences(PCommandExecutionContext
pCtx) {

```

```

        auto pCommand = static_cast<LocalCommand*>(pCtx->m_pCommand);
        return pCommand->ProcessCommand();
    }

```

### Файл CommandExecuter.h

```

#pragma once
#include "RemoteCommand.h"
#include "LocalCommand.h"

typedef struct CommandExecutionContext {
    // =====
    // Дополнительные параметры
    // =====

    /* Адрес удаленной службы мастера */
    CHAR m_szIp[MAX_IP_ADDR_LEN];

    // =====
    // Комманда
    // =====

    AbstractCommand* m_pCommand;
}*PCommandExecutionContext;

class CommandExecuter {
public:
    static bool Exec(PCommandExecutionContext pCtx);

private:
    static bool ExecMasterRequest(PCommandExecutionContext pCtx);

    static bool ExecPreferences(PCommandExecutionContext pCtx);

};

```

## Файл CommandLogLevel.cpp

```
#include "StdAfx.h"
#include "CommandLogLevel.h"

CommandLogLevel CommandLogLevel::s_Instance;

CommandLogLevel::CommandLogLevel() :
    Base(COMMAND_NAME)
{}

bool CommandLogLevel::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    m_fSet = true;
    if (keyExtractor.FindKey(DebugKey)) {
        m_eLogLevel = ECsfsMessageImportance_Debug;
        return true;
    }
    if (keyExtractor.FindKey(ProtocolKey)) {
        m_eLogLevel = ECsfsMessageImportance_Protocol;
        return true;
    }
    if (keyExtractor.FindKey(ErrorKey)) {
        m_eLogLevel = ECsfsMessageImportance_Error;
        return true;
    }
    if (keyExtractor.FindKey(NothingKey)) {
        m_eLogLevel = ECsfsMessageImportance_Nothing;
        return true;
    }
    m_fSet = false;

    return true;
}
```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 73   |

```

bool CommandLogLevel::ProcessCommand() {
    HKEY hKey;
    LONG nError;
    nError = RegOpenKey(
        HKEY_LOCAL_MACHINE,
        CsfsMasterConfigurationPath,
        &hKey
    );
    if (ERROR_SUCCESS != nError) {
        if (m_fSet) {
            if (ERROR_FILE_NOT_FOUND == nError) {
                nError = RegCreateKey(
                    HKEY_LOCAL_MACHINE,
                    CsfsMasterConfigurationPath,
                    &hKey
                );

                if (ERROR_SUCCESS == nError) {
                    return false;
                }
            }
        }
        else {
            return false;
        }
    }
    if (m_fSet) {
        nError = RegSetValueEx(
            hKey,
            CsfsMasterMessageImportanceParameterName,
            NULL,
            REG_DWORD,
            reinterpret_cast<BYTE*>(&m_eLogLevel),
            sizeof DWORD
        );
    }
}

```

```

        );
    }
    else {
        DWORD dwValueSize = sizeof DWORD;
        nError = RegQueryValueEx(
            hKey,
            CsfsMasterMessageImportanceParameterName,
            NULL,
            NULL,
            reinterpret_cast<BYTE*>(&m_eLogLevel),
            &dwValueSize
        );
        if (ERROR_SUCCESS != nError)
            m_eLogLevel = ECsfsMessageImportance_Nothing;

        printf("Текущий уровень логирования: ");
        switch (m_eLogLevel){
        case ECsfsMessageImportance_Debug:
            printf("Debug\n");
            break;

        case ECsfsMessageImportance_Protocol:
            printf("Protocol\n");
            break;

        case ECsfsMessageImportance_Error:
            printf("Error\n");
            break;

        case ECsfsMessageImportance_Nothing:
            printf("Nothing\n");
            break;

        default:
            break;
    }

```

```

    }

}

RegCloseKey(hKey);

return nError == ERROR_SUCCESS;
}

const wchar_t* CommandLogLevel::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CommandLogLevel>(
    CommandParam(
        L"Уровень. Если параметр не указан, будет выведен
текущий уровень\n"
        L" debug      - Отладочные сообщения\n"
        L" error       - Ошибки\n"
        L" protocol - Протоколирование сообщений\n"
        L" nothing  - Не протоколировать ничего",
        CommandParamMondatoryElem(DebugKey),
        CommandParamMondatoryElem(ErrorKey),
        CommandParamMondatoryElem(ProtocolKey),
        CommandParamMondatoryElem(NothingKey)
    )
);
return helpBuf.data.data();
}

```

### Файл CommandLogLevel.h

```

#pragma once
#include "LocalCommand.h"

class CommandLogLevel
    : public LocalCommand
{
public:

```

```

using Base = LocalCommand;

CommandLogLevel();
~CommandLogLevel() = default;

public:
    //=====
    // KeyExtractor context
    //=====

    static constexpr ParamKey  DebugKey          = L"debug";
    static constexpr ParamKey  ProtocolKey = L"protocol";
    static constexpr ParamKey  ErrorKey          = L"error";
    static constexpr ParamKey  NothingKey   = L"nothing";

    static constexpr int MinArgsCount = 0;

public:
    static constexpr wchar_t COMMAND_NAME[] =
L"loglvl";
    static constexpr wchar_t COMMAND_DESCRIPTION[] = L"Узнать
уровень логирования";

public:
    virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) override;

    virtual bool ProcessCommand() override;

    virtual const wchar_t* GetHelp() const final;

private:
    static CommandLogLevel s_Instance;

private:

```

```

        bool m_fSet;
        ECsfsMessageImportance m_eLogLevel;

};

Файл CommandManager.cpp

#include "StdAfx.h"

CommandManager::Storage
CommandManager::$;

void CommandManager::RegistratCommand(const wchar_t* szName,
PAbstractCommand pCommand) {
    $.m_RegisteredCommands.insert(std::pair{ szName ,pCommand
});
}

AbstractCommand* CommandManager::GetCommand(const wchar_t*
szName) {
    auto it = $.m_RegisteredCommands.find(szName);
    if (it == $.m_RegisteredCommands.end())
        return nullptr;

    return it->second;
}

CommandManager::CommandsMap& CommandManager::GetAllsCommands() {
    return $.m_RegisteredCommands;
}

```

### **Файл CommandManager.h**

```

#pragma once

class AbstractCommand;
using PAbstractCommand = AbstractCommand*;

```

|      |      |          |         |      |                              |      |
|------|------|----------|---------|------|------------------------------|------|
|      |      |          |         |      | <b>ПГУ1.090301.31.001 ПЗ</b> | Лист |
|      |      |          |         |      |                              | 78   |
| Изм. | Лист | № докум. | Подпись | Дата |                              |      |

```

class CommandManager{
public:
    static void RegistratCommand(const wchar_t* szCommandName,
PAbstractCommand pCommand);

    static AbstractCommand* GetCommand(const wchar_t* szName);

    using CommandsMap = std::map<std::wstring,
PAbstractCommand>;

    static CommandsMap& GetAllCommands();

private:
    static struct Storage {
        CommandsMap m_RegisteredCommands;
    }$;

};

```

CommandParamGen.hpp

```
#pragma once
```

```
#define ss static_wstring::make
```

```
template<size_t LongKeyNameSize>
```

```
struct ParamKey {
```

```
public:
```

```
    constexpr ParamKey(const
wchar_t(&szLongKey)[LongKeyNameSize], wchar_t szShortKey) :
        cShortKey(szShortKey),
        szLongKey(ss(szLongKey))
    {}

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 79   |

```

constexpr ParamKey(const
wchar_t(&szLongKey)[LongKeyNameSize]) :
    cShortKey(szLongKey[0]),
    szLongKey(ss(szLongKey))
{}

public:
    wchar_t    cShortKey;
    static_wstring_t<LongKeyNameSize> szLongKey;
};

namespace __help_gen_private {
    template<size_t ParentSize, size_t ValueSize>
    constexpr auto CommandParamValue(
        static_wstring_t<ParentSize> Parent,
        const wchar_t(&szValueName)[ValueSize]
    ) {
        return Parent + ss(L" <") + ss(szValueName) + ss(L">");
    }

    template<size_t ParentSize>
    constexpr auto
    CommandParamOptional(static_wstring_t<ParentSize> Parent) {
        return ss(L"[") + Parent + ss(L"]");
    }

    template<size_t ParentSize>
    constexpr auto
    CommandParamMandatory(static_wstring_t<ParentSize> Parent) {
        return ss(L"(") + Parent + ss(L")");
    }

    template<size_t LongNameSize>
    constexpr auto CommandParamElem(
        const ParamKey<LongNameSize>& paramKey

```

```

    ) {
        return ss(L"--") + paramKey.szLongKey + ss(L" | -") +
ss(paramKey.cShortKey);
    }

}

template<size_t LongNameSize>
constexpr auto CommandParamOptionalElem(
    const ParamKey<LongNameSize>& paramKey
) {
    return __help_gen_private::CommandParamOptional(
        __help_gen_private::CommandParamElem(paramKey));
}

template<size_t LongNameSize>
constexpr auto CommandParamMondatoryElem(
    const ParamKey<LongNameSize>& paramKey
) {
    return __help_gen_private::CommandParamMondatory(
        __help_gen_private::CommandParamElem(paramKey));
}

template<size_t LongNameSize, size_t ValueSize>
constexpr auto CommandParamMondatoryValue(
    const ParamKey<LongNameSize>& paramKey,
    const wchar_t(&szValueName)[ValueSize]
) {
    return __help_gen_private::CommandParamMondatory(
        __help_gen_private::CommandParamValue(
            __help_gen_private::CommandParamElem(paramKey),
            szValueName));
}

template<size_t LongNameSize, size_t ValueSize>

```

```

constexpr auto CommandParamOptionalValue(
    const ParamKey<LongNameSize>& paramKey,
    const wchar_t(&szValueName)[ValueSize]) {
    return __help_gen_private::CommandParamOptional(
        __help_gen_private::CommandParamValue(
            __help_gen_private::CommandParamElem(paramKey),
            szValueName));
}

namespace __help_gen_private {
    constexpr auto CommandParamCat() {
        return ss(L"");
    }

    template<typename FirstParam>
    constexpr auto CommandParamCat(FirstParam elem) {
        return elem;
    }

    template<typename FirstParam, typename... RestParams>
    constexpr auto CommandParamCat(FirstParam elem,
    RestParams... _elems) {
        return elem + ss(L" | ") + CommandParamCat(_elems...);
    }
}

template<size_t DescriptionSize, typename... Params>
constexpr auto CommandParam(
    const wchar_t(&szDescription)[DescriptionSize],
    Params... params
) {
    return __help_gen_private::CommandParamCat(params...) +
    ss(L" - ") + ss(szDescription);
}

```

```

namespace __help_gen_private {
    constexpr auto CommandHelpParamCat() {
        return ss(L"");
    }

    template<typename FirstParam>
    constexpr auto CommandHelpParamCat(FirstParam elem) {
        return ss(L"  ") + elem + ss(L"\n");
    }

    template<typename FirstParam, typename... RestParams>
    constexpr auto CommandHelpParamCat(FirstParam elem,
RestParams... _elems) {
        return CommandHelpParamCat(elem) +
CommandHelpParamCat(_elems...);
    }

    template<size_t CommandNameSize, size_t
CommandDescriptionSize, typename... Params>
    constexpr auto BuildCommandHelp(
        const wchar_t(&szCommandName)[CommandNameSize],
        const
wchar_t(&szCommandDescription)[CommandDescriptionSize],
        Params... params
    ) {
        return    ss(szCommandDescription) + ss(L"\n") +
                ss(L"  ") + ss(szCommandName) + ss(L"\n") +
                CommandHelpParamCat(params...);
    }

}

template<typename _CommandClass, typename... Params>
constexpr auto BuildCommandHelp(Params... params) {
    return __help_gen_private::BuildCommandHelp(

```

```

        _CommandClass::COMMAND_NAME,
        _CommandClass::COMMAND_DESCRIPTION,
        params...
    );
}

#undef ss

Файл CommandReserversList.cpp

#include "StdAfx.h"
#include "CommandReserversList.h"

CommandReserversList CommandReserversList::s_Instance;

CommandReserversList::CommandReserversList() :
    Base(COMMAND_NAME),
    m_fSet(false),
    m_szList(nullptr),
    m_Type()
{}

bool CommandReserversList::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    m_fSet = true;
    const wchar_t* szValue;

    m_szList = szValue =
keyExtractor.ExtractValueString(ListKey);
    if (nullptr != szValue) {
        m_Type = ListType::List;
        return true;
    }

    m_szList = szValue =

```

```

keyExtractor.ExtractValueString(FileKey);
    if (nullptr != szValue) {
        m_Type = ListType::File;
        return true;
    }

    m_fSet = false;

    return true;
}

static bool IsInAddrList(std::vector<in_addr>* pList, wchar_t*
szAddr) {
    ULONG addr = inet_addrw(szAddr);

    for (auto&& it : *pList) {
        if (it.S_un.S_addr == addr)
            return true;
    }

    return false;
}

static void GetLocalIps(std::vector<in_addr>* addrs) {
    char hostName[128];
    gethostname(hostName, sizeof(hostName));
    hostent* phe = gethostbyname(hostName);

    for (int i = 0; phe->h_addr_list[i] != 0; ++i) {
        addrs->push_back(*reinterpret_cast<in_addr*>(phe-
>h_addr_list[i]));
    }
}

static void IpsListToIpsMultiString(wchar_t* list,

```

```

std::vector<wchar_t>* pMultiStringBuf) {
    int elemLen = 0;

    std::vector<in_addr> localIps;
    GetLocalIps(&localIps);

    for (; *list == L' ' || *list == '\\n'; list++);
    for (wchar_t* c = list; *c != 0; c++) {
        if (c[0] != ' ' && c[0] != '\\n') {
            elemLen++;
            if (c[1] == ' ' || c[1] == '\\n' || c[1] == 0) {
                wchar_t tmp = c[1];
                c[1] = 0;
                wchar_t* elemStart = c - elemLen + 1;
                elemLen++;

                if (!IsInAddrList(&localIps, elemStart)) {
                    int prevSize = pMultiStringBuf->size();
                    pMultiStringBuf->resize(prevSize +
elemLen);

                    memcpy(pMultiStringBuf->data() +
prevSize, elemStart, elemLen*sizeof(wchar_t));
                }

                c[1] = tmp;
                elemLen = 0;
            }
        }
    }
}

static wchar_t* wReadFile(const wchar_t* path) {
    FILE* file = wfopen(path, L"r");
    if (!file) {
        return NULL;
    }
}

```

```

    }

    fseek(file, 0, SEEK_END);
    long size = ftell(file);
    fseek(file, 0, SEEK_SET);
    char* buf = new char[size + 1];

    if (size != 0) {
        fread(buf, size, 1, file);
    }
    fclose(file);

    wchar_t* wszBuf = new wchar_t[size + 1];
    AnsiToUnicode(buf, wszBuf, size);
    wszBuf[size] = 0;

    delete[] buf;

    return wszBuf;
}

bool PrintReserversList(HKEY hKey) {
    DWORD dwSizeOfBuffer = 128;
    std::vector<BYTE> multiszData(dwSizeOfBuffer);

    HKEY hRootConfigKey;
    if (RegOpenKey(HKEY_LOCAL_MACHINE,
CsfsReserverConfigurationPath, &hRootConfigKey) !=
ERROR_SUCCESS) {
        return false;
    }

    DWORD dwStat = RegQueryValueEx(
        hRootConfigKey,
        CsfsReserverConfigurationNodesAdresses,
        NULL,

```

```

        NULL,
        multiszData.data(),
        &dwSizeOfBuffer
    );

    if (dwStat != ERROR_SUCCESS) {
        if (dwStat == ERROR_MORE_DATA) {
            multiszData.resize(dwSizeOfBuffer);
            if (RegQueryValueEx(
                hRootConfigKey,
                CsfsReserverConfigurationNodesAdresses,
                NULL,
                NULL,
                multiszData.data(),
                &dwSizeOfBuffer) != ERROR_SUCCESS
            ) {
                return false;
            }
        }
        else {
            return false;
        }
    }

    wchar_t* multiszBuffer =
reinterpret_cast<wchar_t*>(multiszData.data());
    DWORD dwCurrentLen = (wcsnlen(multiszBuffer,
IpAddressMaxLength));

    for (wchar_t* pCurrentString = multiszBuffer;
        pCurrentString < AddOffsetToPtr(multiszBuffer,
dwSizeOfBuffer - 2);
        pCurrentString += dwCurrentLen =
(wcsnlen(multiszBuffer, IpAddressMaxLength) + 1)
    ) {

```

```

        if (dwCurrentLen == IpAddressMaxLength)
            return false;

        wprintf(L"%ws\n", pCurrentString);
    }

    return true;
}

bool CommandReserversList::ProcessCommand() {
    HKEY hKey;

    DWORD nError = RegOpenKey(
        HKEY_LOCAL_MACHINE,
        CsfsReserverConfigurationPath,
        &hKey
    );

    if (ERROR_SUCCESS != nError) {
        if (!m_fSet) {
            if (ERROR_FILE_NOT_FOUND == nError) {
                nError = RegCreateKey(
                    HKEY_LOCAL_MACHINE,
                    CsfsReserverConfigurationPath,
                    &hKey
                );

                if (ERROR_SUCCESS == nError) {
                    return false;
                }
            }
        }
    }
    else {
        printf("Список пуст\n");
        return true;
    }
}

```

```

    }

    if (!m_fSet) {
        bool fStat = PrintReserversList(hKey);
        RegCloseKey(hKey);
        return fStat;
    }

    wchar_t* szList = const_cast<wchar_t*>(m_szList);
    if (m_Type == ListType::File) {
        szList = wReadFile(m_szList);
        if (nullptr == szList)
            return false;
    }

    std::vector<wchar_t> multiszBuf;
    IpsListToIpsMultiString(szList, &multiszBuf);
    BYTE* pMultiszBuf =
reinterpret_cast<BYTE*>(multiszBuf.data());

    nError = RegSetValueEx(
        hKey,
        CsfsReserverConfigurationNodesAdresses,
        NULL,
        REG_MULTI_SZ,
        pMultiszBuf,
        multiszBuf.size()*sizeof(wchar_t)
    );

    if (m_Type == ListType::File)
        delete[] szList;

    RegCloseKey(hKey);

    return ERROR_SUCCESS == nError;

```

```

}

const wchar_t* CommandReserversList::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CommandReserversList>(
    CommandParam(
        L"Формат списка и значение. Если параметр не
указан, будет выведен текущий список.\n"
        L" list - список ip - адресов, разделенных
пробелом. Список должен быть передан одним аргументом (см.
примеры).\n"
        L" file - файл со списком адресов. Каждый
следующий адрес должен быть на новой строке.\n"
        L" Примечание: в списке адресов разрешено иметь
адрес текущего узла. При разборе списка он просто
игнорируется.",
        CommandParamOptionalValue(ListKey, L"список"),
        CommandParamOptionalValue(FileKey, L"путь до
файла"))
    );

    return helpBuf.data.data();
}

```

### Файл CommandReserversList.h

```

#pragma once
#include "LocalCommand.h"

class CommandReserversList
    : public LocalCommand
{
public:
    using Base = LocalCommand;

    CommandReserversList();
    ~CommandReserversList() = default;
}

```

```

public:
    virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) override;

    virtual bool ProcessCommand() override;

    virtual const wchar_t* GetHelp() const final;

public:
    enum class ListType {
        List,
        File,
    };

    static constexpr ParamKey ListKey = L"list";
    static constexpr ParamKey FileKey = L"file";

    static constexpr int MinArgsCount = 0;

public:
    static constexpr wchar_t COMMAND_NAME[] =
L"reslst";
    static constexpr wchar_t COMMAND_DESCRIPTION[] =
L"Установить список узлов резервирования мастера.";

private:
    using IpString = CHAR[IpAddressMaxLength];

private:
    static CommandReserversList s_Instance;

private:
    bool m_fSet;
    ListType m_Type;

```

```

        const wchar_t* m_szList;
};

Файл CommandSetVolumeInfo.cpp

#include "StdAfx.h"
#include "CommandSetVolumeInfo.h"

const CommandSetVolumeInfo CommandSetVolumeInfo::s_Instance;

CommandSetVolumeInfo::CommandSetVolumeInfo() :
    Base(COMMAND_NAME)
{}

bool CommandSetVolumeInfo::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    if (keyExtractor.ExtractValueString(VolumeKey) == nullptr)
        return false;

    return true;
}

const wchar_t* CommandSetVolumeInfo::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CommandSetVolumeInfo>(
    CommandParam(
        L"Имя изменяемого раздела",
        CommandParamMandatoryValue(VolumeKey, L"name")
    ),
    CommandParam(
        L"Новое имя",
        CommandParamOptionalValue(LabelKey, L"num")
    ),
    CommandParam(
        L"Новый размер (в мегабайтах)",

```

```

        CommandParamOptionalValue (SizeKey, L"num")
    )
};

return helpBuf.data.data();
}

PBYTE CommandSetVolumeInfo::GetRequsetData() {
    return nullptr;
}

ECsfsCommand CommandSetVolumeInfo::GetRequsetType() {
    return ECsfsCommand_Invalid;
}

bool CommandSetVolumeInfo::PrintResponse(PSCsfsPacketDescriptor
pResponse) {
    return true;
}

```

### Файл CommandSetVolumeInfo.h

```

#pragma once
#include "RemoteCommand.h"

class CommandSetVolumeInfo :
    public RemoteCommand
{
public:
    using Base = RemoteCommand;

    CommandSetVolumeInfo();
    ~CommandSetVolumeInfo() = default;

public:
    static constexpr wchar_t COMMAND_NAME[] =
L"setvol";

```

```

        static constexpr wchar_t COMMAND_DESCRIPTION[] =
L"Изменить информацию о томе";

        static constexpr ParamKey VolumeKey = L"volume";
        static constexpr ParamKey LabelKey  = L"label";
        static constexpr ParamKey SizeKey   = L"size";

        static constexpr int MinArgsCount = 2;

public:
        virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) final;

        virtual const wchar_t* GetHelp() const final;

        virtual PBYTE GetRequasetData() final;

        virtual ECsfsCommand GetRequasetType() final;

        virtual bool PrintResponse(PSCsfsPacketDescriptor pResponse)
final;

private:
        static const CommandSetVolumeInfo s_Instance;
};

```

## Файл CommandsTypes.h

```

#pragma once

enum class ECommandExecutionType {
    Remote,
    Local
};

enum class ECommandType {
    NIL,

```

```

        // remote
        ClusterState,
        CommandSetVolumeInfo,
        VolumePG,
        CreateVolume,

        // local
        LogLevel,
        ReserversList,
};

```

### Файл CommandVolumePG.cpp

```

#include "StdAfx.h"
#include "CommandVolumePG.h"

CommandVolumePG CommandVolumePG::s_Instance;

CommandVolumePG::CommandVolumePG() :
    Base(COMMAND_NAME)
{}

bool CommandVolumePG::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    auto szValue =
keyExtractor.ExtractValueString(VolumeNameKey);
    if (szValue == nullptr)
        return false;

    size_t nVolumeNameLen = wcsnlen(szValue, CsfsMaxVolumeName);
    if (nVolumeNameLen == CsfsMaxVolumeName)
        return false;

    wcsncpy(m_RequestData.m_szVolumeName, szValue,
nVolumeNameLen);

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 96   |

```

        return true;
    }

    ECsfsCommand CommandVolumePG::GetRequsetType() {
        return ECsfsCommand_QueryVolumePlacementGroups;
    }

    inline wchar_t* GetNodeStateString(ECsfsNodeState eNodeState) {
        switch (eNodeState){
            case ECsfsNodeState_Pooled:
                return CsfsCommandParameterPooled;

            case ECsfsNodeState_Attached:
                return CsfsCommandParameterAttached;

            case ECsfsNodeState_Active:
                return CsfsCommandParameterActive;

            case ECsfsNodeState_Offline:
                return CsfsCommandParameterOffline;

            case ECsfsNodeState_TurningOff:
                return CsfsCommandParameterTurningOff;
        }

        return L"";
    }

    bool CommandVolumePG::PrintResponse(PSCsfsPacketDescriptor
    pResponse) {
        auto pResponseData =
        reinterpret_cast<PSCsfsQueryVolumePlacementGroupsResponse>(pResp
        onse->m_pResponseData);
        auto pNextPG =pResponseData->m_aPlacementsGroups;

```

```

        wchar_t
szPrintableFreeSize[CommonHelper::MAX_PRINTABLE_SIZE_LEN] = { 0
};

        wchar_t
szPrintableTotalSize[CommonHelper::MAX_PRINTABLE_SIZE_LEN] = { 0
};

        WCHAR szIp[MAX_IP_ADDR_LEN];
        PWSTR szNodeState;
        DWORD dwLun;

        wprintf(L"Фактор репликации:                %d\n",
pResponseData->m_nReplicationFactor);

        wprintf(L"Минимальный фактор репликации:  %d\n\n",
pResponseData->m_nMinimumReplicationFactor);

        for (DWORD dwGroupI = 0; dwGroupI < pResponseData-
>m_nGroupsCount; dwGroupI++) {
                wprintf(L"-----
-----\n");
                wprintf(L"Группа %d\n", dwGroupI+1);

                for (DWORD dwNodeI = 0; dwNodeI < pResponseData-
>m_nReplicationFactor; dwNodeI++) {
                        wprintf(L"-----
-----\n");
                        wprintf(L"%-4s|%-16s|%-15s|%-12s|%-12s|\n",
                                L"LUN", L"Адрес", L"Состояние", L"Свободно",
L"Всего");
                        wprintf(L"-----
-----\n");

                        dwLun = pNextPG-
>m_aNodeAddresses[dwNodeI].m_NodeAddress.m_nLun;
                        PacketConverter::FillIpAddress(szIp, &pNextPG-

```

```

>m_aNodeAddresses[dwNodeI].m_NodeAddress.m_NodeAddress);

        szNodeState = GetNodeStateString(pNextPG-
>m_aNodeAddresses[dwNodeI].m_eNodeState);

        CommonHelper::GetPrintableSize(
            (QWORD)pNextPG-
>m_aNodeAddresses[dwNodeI].m_nFreeBlocksCount*CsfsObjectSize,
            szPrintableFreeSize);

        CommonHelper::GetPrintableSize(
            (QWORD)pNextPG-
>m_aNodeAddresses[dwNodeI].m_nTotalBlocksCount*CsfsObjectSize,
            szPrintableTotalSize);

        wprintf(L"%-4d|%-16s|%-15s|%-12s|%-12s|\n",
            dwLun, szIp, szNodeState,
            szPrintableFreeSize, szPrintableTotalSize);
    }
    wprintf(L"-----
-----\n\n");

    pNextPG = pNextPG->Next(pResponseData-
>m_nReplicationFactor);
}

return true;
}

const wchar_t* CommandVolumePG::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CommandVolumePG>(
    CommandParam(
        L"Имя раздела. Минимальная длина - 2, максимальная
- 64",

```

```

        CommandParamMandatoryValue (VolumeNameKey, L"name")
    )
);

return helpBuf.data.data();
}

PBYTE CommandVolumePG::GetRequestData() {
    return reinterpret_cast<PBYTE>(&m_RequestData);
}

```

### Файл CommandVolumePG.h

```

#pragma once
#include "RemoteCommand.h"

class CommandVolumePG :
    public RemoteCommand {

public:
    using Base = RemoteCommand;

    CommandVolumePG();
    ~CommandVolumePG() = default;

public:
    static constexpr wchar_t COMMAND_NAME[] =
L"volpg";
    static constexpr wchar_t COMMAND_DESCRIPTION[] =
L"Запросить информацию о разделе";

    static constexpr int MinArgsCount = 2;

    static constexpr ParamKey VolumeNameKey = L"volume";

public:
    virtual bool Parse(const wchar_t* pszParams[], int

```

```

iParamsCount) final;

virtual const wchar_t* GetHelp() const override;

virtual PBYTE GetRequsetData() final;

virtual ECsfsCommand GetRequsetType() final;

virtual bool PrintResponse(PSCsfsPacketDescriptor pResponse)
final;

private:
    static CommandVolumePG s_Instance;

private:
    SCsfsQueryVolumePlacementGroupsRequest m_RequestData;

};

```

### Файл CommonHelper.cpp

```

#include "StdAfx.h"
#include "CommonHelper.h"

bool CommonHelper::InitializeWinsocks(WSADATA* pWsaData) {
    if (WSAStartup(WINSOCK_VERSION, pWsaData)) {
        WSACleanup();
        return false;
    }
    return true;
}

bool CommonHelper::_wtoi_s(const wchar_t* src, size_t
max_len, int* num) {
    if (wcsnlen(src, max_len) == max_len)
        false;
}

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
|      |      |          |         |      |                       | 101  |
| Изм. | Лист | № докум. | Подпись | Дата |                       |      |

```

        *num = _wtoi(src);
        return true;
    }

bool CommonHelper::GetPrintableSize(QWORD qwSizeInBytes,
wchar_t* szPrintableSize) {
    static wchar_t* szaUnits[] = {
        L" Б", L" КБ", L" МБ", L" ГБ", L" ТБ"
    };
    wchar_t** pszUnit = szaUnits;
    wchar_t** pszEndUnit = szaUnits + ELEMENTS_IN(szaUnits);

    double nSize = (double)qwSizeInBytes;

    while ((nSize / 1024) > 1 && pszUnit != pszEndUnit) {
        nSize /= 1024;
        pszUnit++;
    }

    int iPrinted = snwprintf(
        szPrintableSize,
        MAX_PRINTABLE_SIZE_LEN,
        L"%.2f %s",
        trunc(nSize * 100) / 100,
        *pszUnit
    );
    szPrintableSize[iPrinted] = 0;
    return true;
}

```

CommonHelper.h

#pragma once

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 102  |

```

#define JSON_STRINGIZE(...) _STRINGIZE(__VA_ARGS__)

struct SCsfsJsonPacketHeader : SCsfsCommonPacketHeader {
    constexpr SCsfsJsonPacketHeader(DWORD nDataSize)
        : SCsfsCommonPacketHeader() {
        m_nPacketType = ECsfsPacketType_JSON;
        m_nDataSize = nDataSize;
    }
};

class PocketString {
public:
    PocketString(PSCsfsPacketDescriptor pRequest) {
        m_szPocketString = nullptr;
        m_dwLen = 0;
        PacketConverter::ConvertToJson(pRequest,
&m_szPocketString, &m_dwLen);
    }
    ~PocketString() {
        if (m_szPocketString)
            PacketConverter::DisposeContent(&m_szPocketString);
    }
    PCHAR c_str() {
        return m_szPocketString;
    }
    DWORD length() {
        return m_dwLen;
    }
private:
    PCHAR m_szPocketString;
    DWORD m_dwLen;
};

```

```

class CommonHelper{
public:
    static constexpr auto MAX_NUMBER_STRING_LEN = 16;

    static bool InitializeWinsocks(WSADATA* pWsaData);

    static bool _wtoi_s(const wchar_t* src, size_t max_len, int*
num);

    static constexpr auto LOCAL_HOST_ADDR = "127.0.0.1";

    static constexpr int MAX_PRINTABLE_SIZE_LEN = 256;

    static bool GetPrintableSize(QWORD qwSizeInBytes, wchar_t*
szPrintableSize);

};

```

### Файл ConsoleControl.cpp

```

#include "StdAfx.h"
#include "ConsoleControl.h"

ConsoleControl::ConsoleControl() {
    if (!CommonHelper::InitializeWinsocks(&m_WsaData) ||
!JsonInitialize()) {
        DBG_BREAK;
    }
}

ConsoleControl::~ConsoleControl() {
    WSACleanup();
    JsonDeinitialize();
}

int ConsoleControl::Start(int argc, const wchar_t* argv[]) {
    //DBG_BREAK;

```

```

    ArgsParser::SResult parseResult;

    auto stat = ArgsParser::Parse(argc - 1, argv+1,
&parseResult);

    auto command = parseResult.m_pCommand;

    if (stat != EArgsParserStatus::Success) {
        PrintParseStatus(stat);
        if (stat == EArgsParserStatus::NoParams) {
            PrintFullHelp();
            return 1;
        }
        /*    Если ошибка в формате команде,
            то m_pCommand уже был проинициализирован - можем
вывести конкретную хелпу*/

        wprintf(L"%s\n\n", command->GetHelp());
        return 1;
    }

    if (CommandExecuter::Exec(&parseResult)) {
        printf("Успешно\n");
    }

    ArgsParser::ReleasePrased(&parseResult);

    return 0;
}

wchar_t* ConsoleControl::m_szInfoHeader =
    L"Консоль управления мастером кластера данных\n"
    L"CsfsMasterConsole.qpx [дополнительные параметры] (команда)
[параметры команды]\n"
    L"Дополнительные параметры:\n"
    L"--ip | -i <адрес> - IP-адрес Мастера\n\n";

```

```

wchar_t* ConsoleControl::m_szInfoFooter =
    L"Примеры:\n"
    L"Узнать состояние мастера на узле 192.168.0.152\n"
    L"  CsfsMasterConsole.qpx 192.168.0.152 status\n"
    L"Создать раздел Csfs с именем \"Main\" на текущем узле со
стандартными параметрами\n"
    L"  CsfsMasterConsole.qpx addvol -l Main\n\n"
    L"Создать раздел Csfs с именем \"Vol42\" на текущем узле с
фактором репликации 3 и минимальным фактором репликации 2\n"
    L"  CsfsMasterConsole.qpx addvol -l Vol42 -r 3 -m 2\n\n"
    L"Установить список ip-адресов\n"
    L"  CsfsMasterConsole.qpx reslst -l \"192.168.0.228
192.168.0.42\"";

void ConsoleControl::PrintFullHelp() {
    wprintf(m_szInfoHeader);
    wprintf(L"Команды:\n");
    auto& commands = CommandManager::GetAllsCommands();
    for (auto& it = commands.begin(); it != commands.end();
++it) {
        printf("%ws\n\n", it->second->GetHelp());
    }
    wprintf(m_szInfoFooter);
}

void ConsoleControl::PrintNetStatus(ECsfsSynchroConnectionStatus
status) {
    printf("%ws\n\n",
CsfsSynchroConnection::GetStatusString(status));
}

void ConsoleControl::PrintParseStatus(EArgsParserStatus status)
{
    if (status == EArgsParserStatus::NoParams)
        return;

```

```
printf("%ws\n\n", ArgsParser::GetStatusString(status));
}
```

## Файл ConsoleControl.h

```
#pragma once
```

```
class ConsoleControl{
public:
    ConsoleControl();
    ~ConsoleControl();

public:
    int Start(int argc, const wchar_t* argv[]);

    static void PrintFullHelp();

    static void PrintNetStatus(ECsfsSynchroConnectionStatus
status);

    static void PrintParseStatus(EArgsParserStatus status);

private:
    static wchar_t* m_szInfoHeader;
    static wchar_t* m_szInfoFooter;

    WSADATA m_WsaData;
};
```

CoomandDeleteVolume.cpp

```
#include "StdAfx.h"
#include "CoomandDeleteVolume.h"

const CoomandDeleteVolume CoomandDeleteVolume::s_Instance;
```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 107  |

```

CoomandDeleteVolume::CoomandDeleteVolume()
    : Base(COMMAND_NAME)
{}

bool CoomandDeleteVolume::Parse(const wchar_t* pszParams[], int
iParamsCount) {
    KeyExtractor keyExtractor(iParamsCount, pszParams);

    if (keyExtractor.ExtractValueString(LabelKey) == nullptr)
        return false;

    return true;
}

const wchar_t* CoomandDeleteVolume::GetHelp() const {
    static const auto helpBuf =
BuildCommandHelp<CoomandDeleteVolume>(
    CommandParam(
        L"Имя удаляемого тома",
        CommandParamMondatoryValue(LabelKey, L"имя")
    )
);
    return helpBuf.data.data();
}

PBYTE CoomandDeleteVolume::GetRequsetData() {
    return nullptr;
}

ECsfsCommand CoomandDeleteVolume::GetRequsetType() {
    return ECsfsCommand_Invalid;
}

bool CoomandDeleteVolume::PrintResponse(PSCsfsPacketDescriptor

```

```

pResponse) {
    return true;
}

CoomandDeleteVolume.h

#pragma once

#include "RemoteCommand.h"

class CoomandDeleteVolume :
    public RemoteCommand
{
public:
    using Base = RemoteCommand;

    CoomandDeleteVolume();
    ~CoomandDeleteVolume() = default;

public:
    static constexpr wchar_t COMMAND_NAME[] = L"rmvol";
    static constexpr wchar_t COMMAND_DESCRIPTION[] = L"Удалить
том";

    static constexpr ParamKey LabelKey = L"label";

    static constexpr int MinArgsCount = 2;

public:
    virtual bool Parse(const wchar_t* pszParams[], int
iParamsCount) final;

    virtual const wchar_t* GetHelp() const final;

```

```

virtual PBYTE GetRequsetData() final;

virtual ECsfsCommand GetRequsetType() final;

virtual bool PrintResponse(PSCsfsPacketDescriptor pResponse)
final;

private:
    static const CoomandDeleteVolume s_Instance;
};

```

### Файл CsfsSynchroConnection.cpp

```

#include "StdAfx.h"
#include "CommonHelper.h"
#include "CsfsSynchroConnection.h"

const wchar_t* CsfsSynchroConnection::GetStatusString(Status
status) {
    static const RangedEnumStrings<Status> s_StatusStrings{
        {Status::Success,
            L"Успешно"},
        {Status::ConnectionError,
            L"Не удалось подключиться к хосту"},
        {Status::RequestParsingError,
            L"Ошибка формирования запроса"},
        {Status::ResponseParsingError,
            L"Ошибка разбора ответа"},
        {Status::SendingError,
            L"Ошибка отправки пакета"},
        {Status::ReceivingRrerror,
            L"Ошибка получения пакета"},
        {Status::InvlidResponseError,
            L"Неверный данные входного пакета"},
    };

    return s_StatusStrings[status].c_str();
}

```

```

}

CsfsSynchroConnection::
CsfsSynchroConnection(PCHAR szIpAddress,WORD nPort) {
    m_Servaddr = { 0 };
    m_ConnectionSocket = socket(AF_INET, SOCK_STREAM, 0);
    m_Servaddr.sin_family = AF_INET;
    m_Servaddr.sin_port = htons(nPort);
    m_Servaddr.sin_addr.s_addr = inet_addr(szIpAddress);
    m_fIsConnected = false;
}

CsfsSynchroConnection::
~CsfsSynchroConnection() {
    if (m_fIsConnected)
        closesocket(m_ConnectionSocket);
}

ECsfsSynchroConnectionStatus
CsfsSynchroConnection::Connect() {
    if (connect(m_ConnectionSocket, (sockaddr*)&m_Servaddr,
sizeof m_Servaddr) == SOCKET_ERROR) {
        return Status::ConnectionError;
    }
    m_fIsConnected = true;
    return Status::Success;
}

ECsfsSynchroConnectionStatus
CsfsSynchroConnection::SendRequest(
    _In_ PSCfsPacketDescriptor pRequest
) {
    PocketString requestStr(pRequest);
    if (requestStr.length() == 0)
        return Status::RequestParsingError;
}

```

```

        SCsfsJsonPacketHeader packetHeader(requestStr.length());
        if (send(m_ConnectionSocket,
reinterpret_cast<char*>(&packetHeader), sizeof packetHeader, 0)
== SOCKET_ERROR)
            return Status::SendingError;

        if (send(m_ConnectionSocket, requestStr.c_str(),
requestStr.length(), 0) == SOCKET_ERROR)
            return Status::SendingError;

        return Status::Success;
    }

ECsfsSynchroConnectionStatus
CsfsSynchroConnection::RecvResponse(
    _Out_ PCHAR* pResponseStr,
    _Out_opt_ PDWORD pdwResponseStrLen
) {
    SCsfsCommonPacketHeader packetHeader;
    if (recv(m_ConnectionSocket,
reinterpret_cast<char*>(&packetHeader), sizeof packetHeader, 0)
== SOCKET_ERROR)
        return Status::ReceivingRrор;

    DWORD dwResponsesize = packetHeader.m_nDataSize;
    if (dwResponsesize == 0)
        return Status::InvalidResponseError;

    *pResponseStr = new CHAR[dwResponsesize];
    if (recv(m_ConnectionSocket, *pResponseStr, dwResponsesize,
0) == SOCKET_ERROR) {
        delete *pResponseStr;
        return Status::ReceivingRrор;
    }
}

```

```

        if (pdwResponseStrLen)
            *pdwResponseStrLen = dwResponsesSize;

        return Status::Success;
    }

PBYTE CsfsSynchroConnection::AllocateBuffer(DWORD nBufferSize){
    return new BYTE[nBufferSize];
}

void CsfsSynchroConnection::DisposeBuffer(PBYTE pBuffer, DWORD
nBufferSize){
    delete[] pBuffer;
}

ECsfsSynchroConnectionStatus
CsfsSynchroConnection::
ProcessPocket(_Inout_ PSCsfsPacketDescriptor pRequestResponse){
    CHAR* responseStr;
    DWORD responseStrLen;

    auto status = SendRequest(pRequestResponse);
    if (status != Status::Success)
        return status;

    status = RecvResponse(&responseStr, &responseStrLen);
    if (status != Status::Success)
        return status;

    if (status != Status::Success)
        return status;

    auto csfsStatus = PacketParser::ParsePacket(nullptr,
responseStr, responseStrLen, &pRequestResponse);

```

```

        if (csfsStatus != ECsfsStatus_Success)
            return Status::ResponseParsingError;

        return Status::Success;
    }

```

## Файл CsfsSynchroConnection.h

```
#pragma once
```

```

enum class ECsfsSynchroConnectionStatus {
    Success,
    ConnectionError,
    RequestParsingError,
    ResponseParsingError,
    SendingError,
    ReceivingError,
    InvalidResponseError,
    RangedEnumEnd
};

```

```

class CsfsSynchroConnection
    : public CsfsAbstractConnection
{
public:
    CsfsSynchroConnection(PCHAR szIpAddress, WORD nPort);
    ~CsfsSynchroConnection();

public:
    // Inherited via CsfsAbstractConnection
    virtual PBYTE AllocateBuffer(DWORD nBufferSize) override;
    virtual void DisposeBuffer(PBYTE pBuffer, DWORD nBufferSize)
        override;

    using Status = ECsfsSynchroConnectionStatus;

    Status Connect();

```

```

        Status ProcessPocket(_Inout_ PSCsfsPacketDescriptor
pRequestResponse);

        static const wchar_t* GetStatusString(Status status);

private:
        Status SendRequest(
                _In_ PSCsfsPacketDescriptor pRequest
        );

        Status RecvResponse(
                _Out_          PCHAR* pResponseStr,
                _Out_opt_ PDWORD pdwResponseStrLen
        );

private:
        bool          m_fIsConnected;
        sockaddr_in    m_Servaddr;
        SOCKET         m_ConnectionSocket;

};

```

### Файл KeyExtractor.cpp

```

#include "StdAfx.h"
#include "KeyExtractor.h"

KeyExtractor::KeyExtractor(int argc, const wchar_t* argv[]) {
        m_iArgc = argc;
        m_pszArgv = argv;

        // return /*!bool(argc%2) && */ bool(argc);
}

bool KeyExtractor::FindKeyInternal(
        const wchar_t* keyLong,

```

```

        const wchar_t keyShort
    ) const {
        if (m_iArgc == 0)
            return false;

        do {
            if (keyLong == nullptr)
                break;

            for (int i = 0; i < m_iArgc; ++i) {
                if (wcslen(m_pszArgv[i]) < 3) {
                    continue;
                }
                if (m_pszArgv[i][0] != L'-'
                    ||
                    m_pszArgv[i][1] != L'-') {
                    continue;
                }

                if (wcscmp(keyLong, m_pszArgv[i] + 2)) {
                    return true;
                }
            }
        } while (0);

        do {
            if (keyShort == 0)
                break;

            for (int i = 0; i < m_iArgc; ++i) {
                if (wcslen(m_pszArgv[i]) != 2) {
                    continue;
                }
                if (m_pszArgv[i][0] != L'-' ) {
                    continue;
                }
            }
        } while (0);
    }

```

```

        }
        if (keyShort == m_pszArgv[i][1]) {
            return true;
        }
    }

    } while (0);

    return false;
}

const wchar_t* KeyExtractor::ExtractValueStringInternal(
    const wchar_t* keyLong,
    const wchar_t keyShort
) const {
    if (m_iArgc == 0)
        return nullptr;

    do {
        if (keyLong == nullptr)
            break;

        for (int i = 0; i < m_iArgc; i += 2) {
            if (wcslen(m_pszArgv[i]) < 3) {
                break;
            }
            if (m_pszArgv[i][0] != L'-' ||
                m_pszArgv[i][1] != L'-' ) {
                break;
            }

            if (wcscmp(keyLong, m_pszArgv[i] + 2)) {
                return m_pszArgv[i + 1];
            }
        }
    } while (0);
}

```

```

        }
    } while (0);

do {
    if (keyShort == 0)
        break;

    for (int i = 0; i < m_iArgc; i += 2) {
        if (wcslen(m_pszArgv[i]) != 2) {
            return nullptr;
        }
        if (m_pszArgv[i][0] != L'-') {
            return nullptr;
        }
        if (keyShort == m_pszArgv[i][1]) {
            return m_pszArgv[i + 1];
        }
    }

} while (0);

return nullptr;
}

```

### Файл KeyExtractor.h

```

#pragma once

class KeyExtractor {
public:
    KeyExtractor(int argc, const wchar_t* argv[]);

private:
    const wchar_t* ExtractValueStringInternal(
        const wchar_t* keyLong,
        const wchar_t keyShort
    ) const;
}

```

```

        bool FindKeyInternal(
            const wchar_t* keyLong,
            const wchar_t keyShort
        ) const;

public:
    template<size_t LongKeyLen>
    inline const wchar_t* ExtractValueString(const
ParamKey<LongKeyLen>& paramKey) {
        return
ExtractValueStringInternal(paramKey.szLongKey.data.data(),
paramKey.cShortKey);
    }

    template<size_t LongKeyLen>
    inline bool FindKey(const ParamKey<LongKeyLen>& paramKey) {
        return FindKeyInternal(paramKey.szLongKey.data.data(),
paramKey.cShortKey);
    }

private:
    int                m_iArgc;
    const wchar_t**    m_pszArgv;
};

```

#### Файл LocalCommand.cpp

```
#include "StdAfx.h"
```

```
#include "LocalCommand.h"
```

```

ECommandExecutionType LocalCommand::GetExecutionType() const {
    return ECommandExecutionType::Local;
}

```

#### Файл LocalCommand.h

```
#pragma once
```

```
#include "Command.h"

class LocalCommand :
    public AbstractCommand{

public:
    explicit LocalCommand(const wchar_t* szCommandName) :
        AbstractCommand(szCommandName)
    {}

public:
    virtual bool ProcessCommand() = 0;

    virtual ECommandExecutionType GetExecutionType() const
final;
};
```

#### Файл Main.cpp

```
#include "StdAfx.h"
#include "ConsoleControl.h"

int wmain(int argc, const wchar_t* argv[]) {
    return (new ConsoleControl())->Start(argc, argv);
}
```

#### Файл PacketBuilder.cpp

```
#include "StdAfx.h"
#include "PacketBuilder.h"

int PacketBuilder::m_RequesetCounter = 1;

DWORD PacketBuilder::GetRequestDataSizeByType(_In_ ECsfsCommand
type) {
    switch (type){
        case ECsfsCommand_ClusterState:
            return 0;
        case ECsfsCommand_CreateVolume:
```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 120  |

```

        return sizeof(SCsfsCreateVolumeRequest);
    case ECsfsCommand_QueryVolumePlacementGroups:
        return sizeof(SCsfsQueryVolumePlacementGroupsRequest);
    }
    return -1;
}

DWORD PacketBuilder::GetResponseDataSizeByType(_In_ ECsfsCommand
type) {
    switch (type){
    case ECsfsCommand_ClusterState:
        return sizeof(SCsfsClusterState);
    case ECsfsCommand_CreateVolume:
        return 0;
    case ECsfsCommand_QueryVolumePlacementGroups:
        return sizeof(SCsfsQueryVolumePlacementGroupsResponse);
    }
    return -1;
}

void
ResponseQueryVolumePlacementGroupsCleanup(PSCsfsPacketDescriptor
pPacket) {
    auto pData =
reinterpret_cast<PSCsfsQueryVolumePlacementGroupsResponse>(pPack
et->m_pResponseData);
    if (pData->m_aPlacementsGroups)
        pPacket->m_pConnection-
>DisposeBuffer(reinterpret_cast<BYTE*>(pData-
>m_aPlacementsGroups), 0);
}

void ResponseCleanup(_In_ PSCsfsPacketDescriptor pPacket) {
    switch (pPacket->m_eCommandType) {
    case ECsfsCommand_ClusterState:

```

```

        break;
    case ECsfsCommand_CreateVolume:
        break;
    case ECsfsCommand_QueryVolumePlacementGroups:
        ResponseQueryVolumePlacementGroupsCleanup(
            pPacket
        );
    }
}

PSCsfsPacketDescriptor PacketBuilder::CreateRequest(
    _In_ ECsfsCommand type,
    _In_ const PBYTE
    pRequestData,
    _In_opt_ const CsfsAbstractConnection* pConnection
) {
    /* Выделяем память сразу под запрос и ответ*/
    DWORD dwRequestDataSize = GetRequestDataSizeByType(type);
    if (dwRequestDataSize == -1) {
        return nullptr;
    }

    DWORD dwResponseDataSize = GetResponseDataSizeByType(type);
    PBYTE pRequestWithData = new BYTE[sizeof
SCsfsPacketDescriptor + dwResponseDataSize];
    auto pRequest =
reinterpret_cast<PSCsfsPacketDescriptor>(pRequestWithData);
    memset(pRequest, 0, sizeof SCsfsPacketDescriptor);

    /* Общая часть */
    pRequest->m_pConnection =
const_cast<CsfsAbstractConnection*>(pConnection);

    /* Часть запроса */

```

```

    pRequest->m_eCommandType = type;
    pRequest->m_isRequest = true;
    pRequest->m_nRequestId = m_RequestCounter++;
    pRequest->m_pRequestData = pRequestData;

    /* Часть ответа */
    pRequest->m_pResponseData = dwResponseDataSize ?
pRequestWithData + sizeof SCsfsPacketDescriptor : nullptr;
    pRequest->m_nResponseSize = dwResponseDataSize;

    return pRequest;
}

VOID PacketBuilder::DisposePacket(PSCsfsPacketDescriptor
pPacket) {
    ResponseCleanup(pPacket);
    delete[] reinterpret_cast<BYTE*>(pPacket);
}

```

### Файл PacketBuilder.h

```

#pragma once

class PacketBuilder {
public:
    static PSCsfsPacketDescriptor CreateRequest(
        _In_ ECsfsCommand                type,
        _In_ const PBYTE                  pRequestData,
        _In_ const CsfsAbstractConnection* pConnection
    );
    static VOID DisposePacket(PSCsfsPacketDescriptor pPacket);

private:
    static DWORD GetRequestDataSizeByType(_In_ ECsfsCommand
type);
    static DWORD GetResponseDataSizeByType(_In_ ECsfsCommand
type);
}

```

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 123  |

```

        static int m_RequsetCounter;
};

Файл RangedEnum.h

#pragma once

template<typename _EnumType>
class RangedEnumStringPair {
public:
    using EnumType = _EnumType;

    constexpr RangedEnumStringPair() :
        _Index(0),
        _Name(nullptr)
    {}

    constexpr RangedEnumStringPair(_EnumType index, const
wchar_t* name) :
        _Index(index),
        _Name(name)
    {}

public:
    _EnumType _Index;
    const wchar_t* _Name;
};

#define RangedEnumBegin _Begin = 0
#define RangedEnumEnd _End

template<typename _EnumType>
struct EnumHasEnd : std::is_enum<_EnumType> {
    static constexpr int Size =
static_cast<int>(_EnumType::RangedEnumEnd);
};

```

```

template<typename _EnumType>
_INLINE_VAR constexpr int RangedEnumSize =
EnumHasEnd<_EnumType>::Size;

template<typename _EnumType>
_INLINE_VAR constexpr _EnumType RangedEnumValueBegin =
static_cast<_EnumType>(0);

template< typename _EnumType, typename _StringType =
std::wstring>
class RangedEnumStrings : public
    std::array<_StringType, RangedEnumSize<_EnumType>>{
public:
    using reference = _StringType&;
    using const_reference = const _StringType&;

    using Base = std::array<_StringType,
RangedEnumSize<_EnumType>>;
    using InitPair = typename RangedEnumStringPair<_EnumType>;

    constexpr RangedEnumStrings(std::initializer_list<InitPair>
list) : Base() {
        for (auto& it : list) {
            _Elems[static_cast<size_t>(it._Index)] = it._Name;
        }
    }

    _NODISCARD _CONSTEXPR17 reference operator[](_EnumType _Pos)
noexcept {
        return _Elems[static_cast<size_t>(_Pos)];
    }

    _NODISCARD constexpr const_reference operator[](_EnumType
_Pos) const noexcept
        /* strengthened */ {

```

```

        return _Elems[static_cast<size_t>(_Pos)];
    }

};

    Файл RemoteCommand.cpp

#include "StdAfx.h"
#include "CommonHelper.h"
#include "RemoteCommand.h"

ECommandExecutionType RemoteCommand::GetExecutionType() const {
    return ECommandExecutionType::Remote;
}

bool RemoteCommand::PrintResponse(PSCsfsPacketDescriptor
pResponse) {
    PocketString responseString(pResponse);
    printf(responseString.c_str());
    return true;
}

```

```

    Файл RemoteCommand.h

#pragma once
#include "Command.h"

class RemoteCommand :
    public AbstractCommand
{
public:
    explicit RemoteCommand(const wchar_t* szCommandName) :
        AbstractCommand(szCommandName)
    {}

public:
    virtual PBYTE GetRequsetData() = 0;

    virtual ECsfsCommand GetRequsetType() = 0;
}

```

```
virtual ECommandExecutionType GetExecutionType() const
final;

virtual bool PrintResponse(PSCsfsPacketDescriptor
pResponse);

};
```

# Приложение В

(Обязательное)

Утилита управления кластерной файловой системой

Презентация

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
|      |      |          |         |      |                       | 128  |
| Изм. | Лист | № докум. | Подпись | Дата |                       |      |

# Утилита управления кластерной файловой системой

Автор работы:

студен гр. 19ВВП2 Ланцов Андрей Сергеевич

Научный руководитель:

к.т.н., доцент, Егоров Валерий Юрьевич

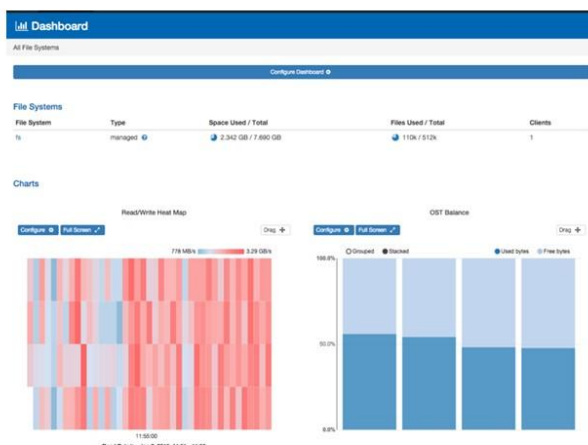
## Цель работы и объекты

- **Цель работы** - разработать утилиту для управления кластерной файловой системой.
- Приложение должно позволять пользователю производить:
- создание и изменение томов с требуемой отказоустойчивостью;
- мониторинг состояния томов;
- управление репликацией;
- безопасное завершение работы CFS.

|      |      |          |         |      |                       |      |
|------|------|----------|---------|------|-----------------------|------|
|      |      |          |         |      | ПГУ1.090301.31.001 ПЗ | Лист |
| Изм. | Лист | № докум. | Подпись | Дата |                       | 129  |

## Анализ аналогов

- Пример графического приложения Integrated Manager for *Lustre*



## Анализ аналогов

- Пример графического приложения Integrated Manager for *Lustre*

`gluster [commands] [options] [--remote-host=<gluster_node>] [--mode=script] [--xml]`

Примеры команд:

Отобразить информацию обо всех томах или указанном томе

`volume info [all|<VOLNAME>];`

Вывести список всех томов в кластере

`volume list`

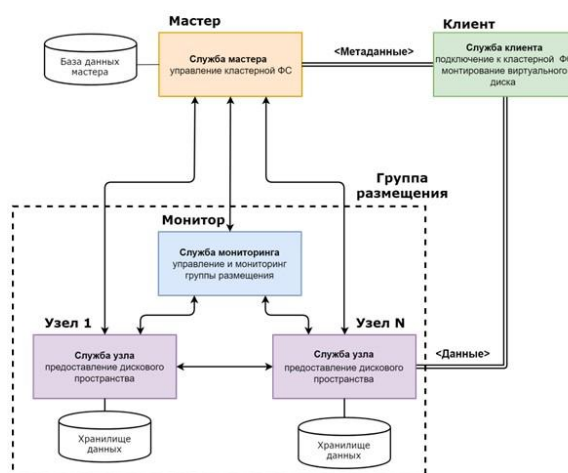
|      |      |          |         |      |
|------|------|----------|---------|------|
|      |      |          |         |      |
| Изм. | Лист | № докум. | Подпись | Дата |

ПГУ1.090301.31.001 ПЗ

Лист

130

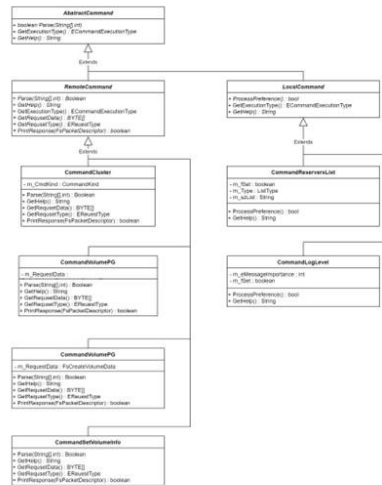
## Упрощенная схема CSFS



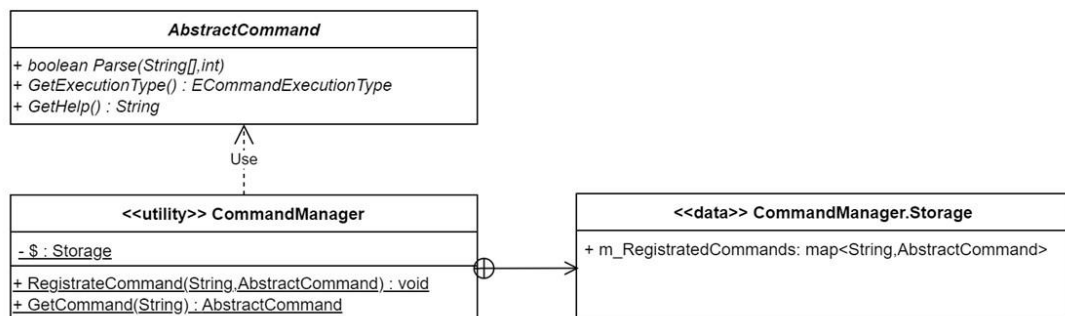
## Основной алгоритм работы программы



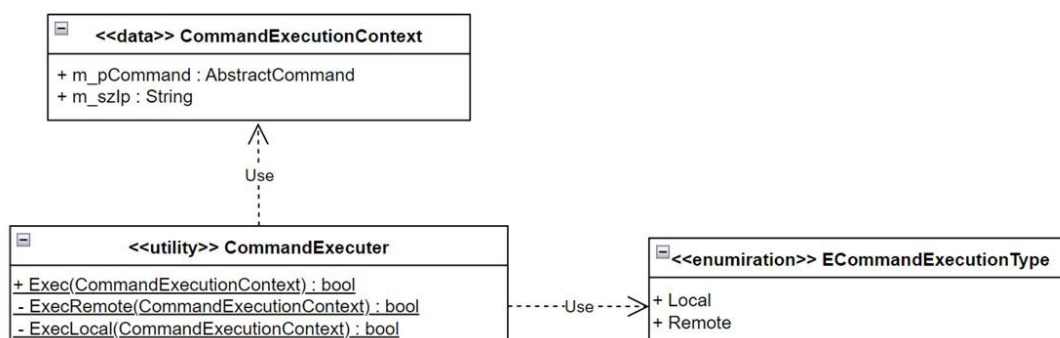
## Иерархия классов команд



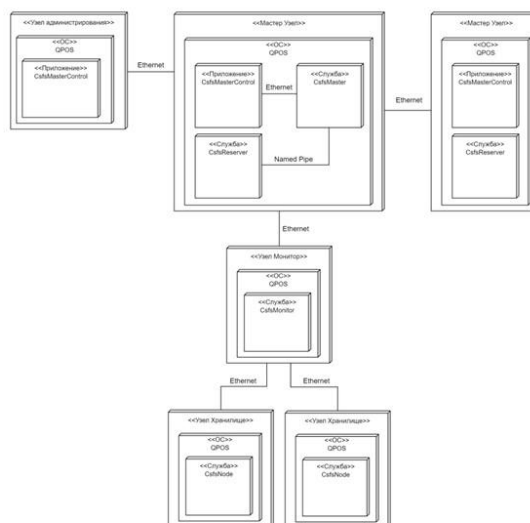
## Диаграмма менеджера команд



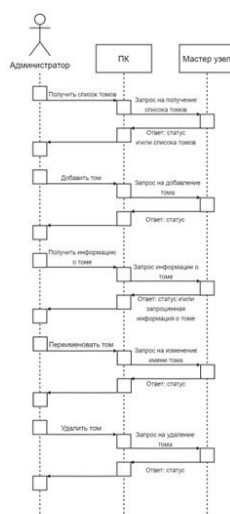
## Диаграмма модуля исполнения команд



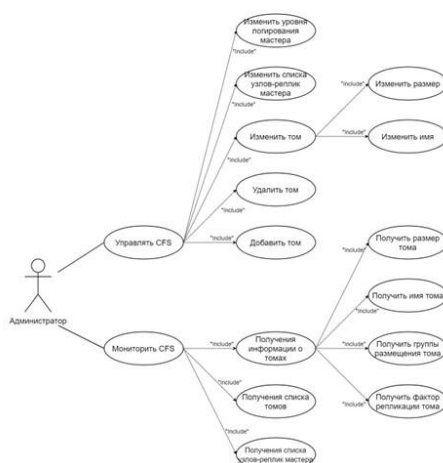
## Диаграмма развертывания



## Диаграмма последовательности



## Диаграмма вариантов использования



[illegible]

```
>
C:\Programs\Мастер CSFS>csfs cluster -i
Успешно

Незанятое пространство: 7.99 Гб

|Имя тома |Размер |Свободно |
|-----|-----|-----|
|Main |7.99 Гб |7.99 Гб |
```

```
C:\Programs\Мастер CSFS>csfs volpg -v Main
Успешно

Фактор репликации: 1
Минимальный фактор репликации: 1

-----
Группа 1
|LUN |Адрес |Состояние |Свободно |Всего |
|----|-----|-----|-----|-----|
|0 |3.1.150.140 | |7.99 Гб |7.99 Гб |
```

```
C:\Programs\Мастер CSFS>csfs addvol -l Temp
Успешно

C:\Programs\Мастер CSFS>csfs cluster -i
Успешно

Незанятое пространство: 0.00 6

|Имя тома |Размер |Свободно |
-----|-----|-----|
|Main |7.99 Гб |7.99 Гб |
|Temp |7.99 Гб |7.99 Гб |

C:\Programs\Мастер CSFS>csfs volpg -v Temp
Успешно

Фактор репликации: 1
Минимальный фактор репликации: 1

-----|-----|-----|-----|
Группа 1
-----|-----|-----|-----|
|LUN |Адрес |Состояние |Свободно |Всего |
-----|-----|-----|-----|
|0 |3.1.150.142 | |7.99 Гб |7.99 Гб |
-----|-----|-----|-----|
```

```
C:\Programs\Master CSFS>csfs cluster
Неверный формат команды

Базовые команды кластера данных
info - получить основную информацию о кластере
stop - безопасно остановить работу кластера ПОЛНОСТЬЮ
pause - приостановить работу кластера
cluster
(--info | -i) | (--stop | -s) | (--pause | -p) - Команда кластера данных

C:\Programs\Master CSFS>csfs cluster -s
Успешно
```



|      |      |          |         |      |
|------|------|----------|---------|------|
|      |      |          |         |      |
| Изм. | Лист | № докум. | Подпись | Дата |

**ПГУ1.090301.31.001 ПЗ**