Министерство образования и науки Российской Федерации

Новосибирский государственный технический университет

Факультет прикладной математики и информатики

ОТЧЁТ

по программе повышения квалификации

Практика коллективной разработки программного  
обеспечения в проектах с открытым кодом.

Основы работы в операционной системе OpenScaler

Группа: ПМИ-12  
Бригада: 1

Студенты: Михайловский М.А.

Швадченко А.В.

Новосибирск, 2023

Оглавление

[Лабораторная работа номер 1 Командный интерфейс Linux 3](#_Toc136347164)

[Лабораторная работа номер 2 Командные сценарии Linux 11](#_Toc136347165)

[Лабораторная работа номер 4 Файловая система OC Linux 13](#_Toc136347166)

[Лабораторная работа номер 5 Системы контроля версий 23](#_Toc136347167)

[Лабораторная работа номер 6 Разработка программ в среде Linux 42](#_Toc136347168)

# Лабораторная работа номер 1 Командный интерфейс Linux

1. Цель работы:

Основным интерфейсом для любой операционной системы (ОС) является командный интерфейс, представленный набором команд. Количество команд ОС может быть достаточно большим, каждая команда реализует одно действие над заданным ресурсом, а основным устройством управления при этом является клавиатура. Графический интерфейс является надстройкой над командным интерфейсом, т.е. любое действие, заданное мышью, выполняется с помощью соответствующей команды ОС.

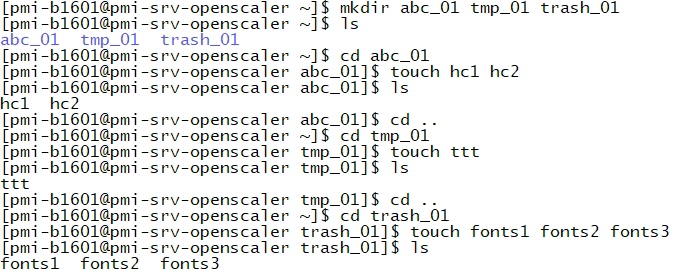
Командный интерфейс требует от пользователя более глубоких знаний устройства компьютера и ОС, поэтому он прежде всего предназначен для применения IT-специалистами. Целью лабораторной работы является приобретение практических навыков работы с интерфейсом командной строки ОС Linux.

1. Выполнение работы:
2. Выполнили вход в систему



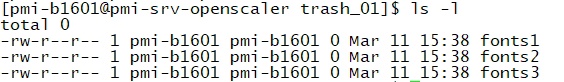
1. Определили полное имя домашнего каталога с помощью команды **pwd**



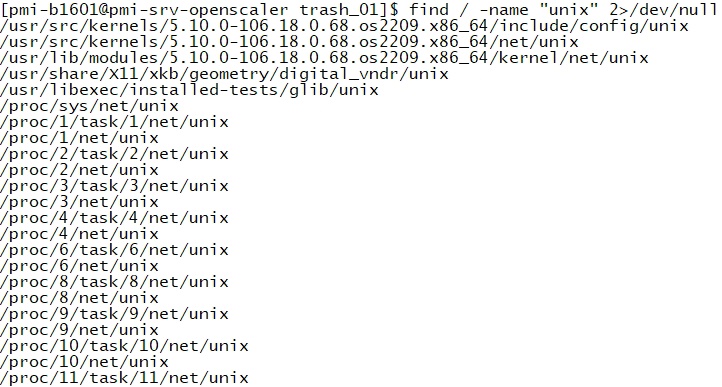
1. При помощи команд **mkdir** и **touch** построили иерархическую структуру файловой системы, показанной на рисунке 1. Команда **mkdir имя\_каталога** создаёт каталог, команда **touch [-опции] имя\_файла** создаёт пустой файл
2. Перешли в каталог trash\_01 с помощью команды **cd [имя\_каталога]**



1. Посмотрели содержимое каталога trash\_01, используя команду **ls –l**, где **l –** опция для получения подробной информации о файлах и каталогах



1. Проверили существование файла unix в корневом каталоге с помощью команды **find начальная\_точка\_поиска [-опции]**



1. Проверили существование подкаталога cron в каталоге /var/spool при помощи команды **find начальная\_точка\_поиска [-опции]**



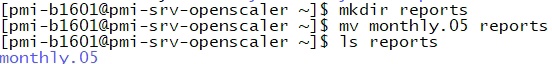
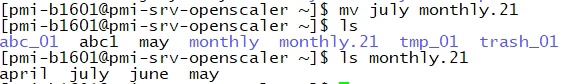
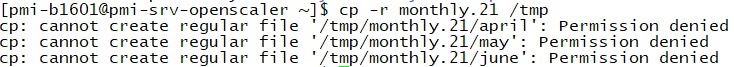
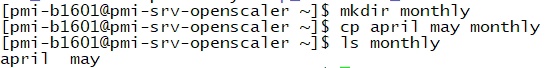
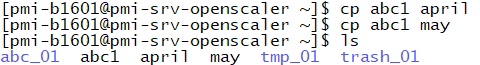
1. Просмотрели содержимое в нашем домашнем каталоге при помощи команды **ls –l**

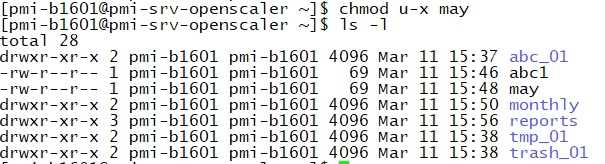


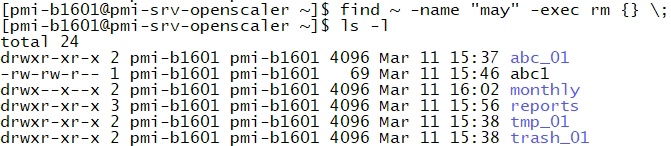
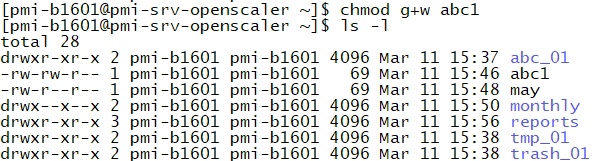
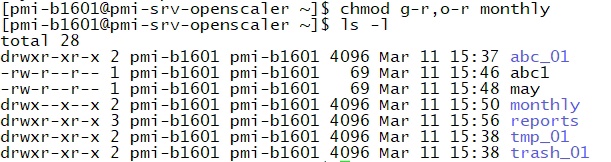
1. При помощи команды **cat имя\_файла** создали файл abc1 и записали в него информацию



1. Выполнили все примеры из 2.3







C:\Users\artem\AppData\Local\Microsoft\Windows\INetCache\Content.Word\9jVFRvjTTug.jpg



11-12. Создаем и удаляем три каталога с помощью команд **mkdir имя\_каталога1 [имя\_каталога2...]** и команды **rm –rf** соответственно

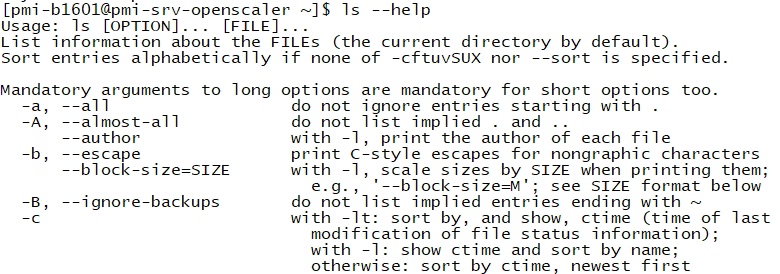
C:\Users\artem\AppData\Local\Microsoft\Windows\INetCache\Content.Word\mACDXiWgyV4.jpg



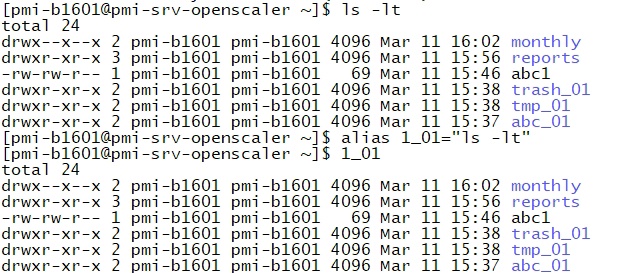
13. Удаляем каталог командой **rm**. Без использования опции **–rf** удалить каталогнельзя, так как эта команда предназначена для удаления файлов



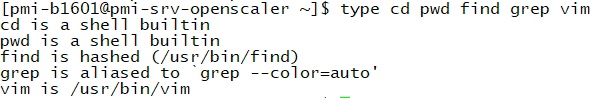
14. Просмотр каталога с его подкаталогами. С помощью команды **команда --help** узнали необходимую опцию для команды **ls [-опции] [путь],** чтобы посмотреть каталог с его подкаталогами



15. Просмотр с сортировкой по времени модификации. С помощью команды **команда –help** определили набор опций команды **ls [-опции] [путь]**. С помощью команды **alias новое\_имя=”команда”** задали псевдоним **1\_01** команде **ls -lt**



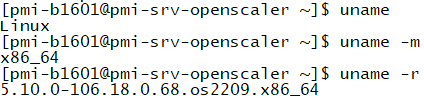
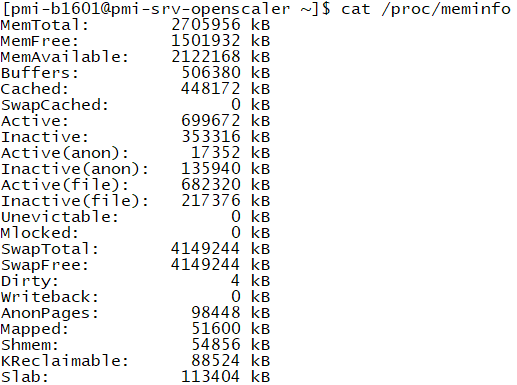
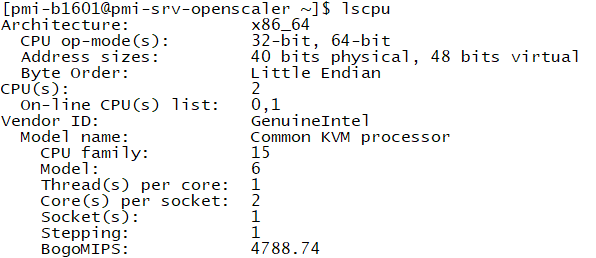
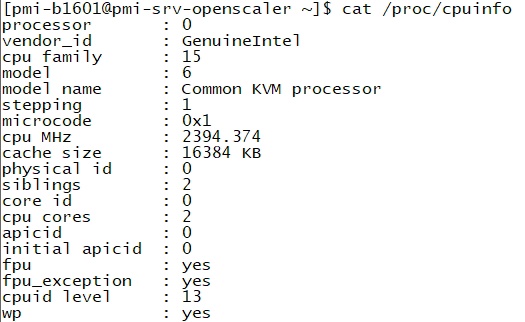
16. Определили типы команд с помощью команды **type команда**



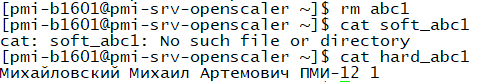
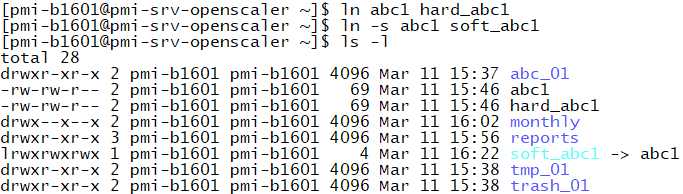
17. При помощи конвейера команд определили количество пользователей на сервере. Команда **who** выводит список активных пользователей, команда **wc -l** выводит число строк в файле. Разделитель команд “|” нужен для того, чтобы выходные данные команды **who** были поданы на вход команды **wc -l**



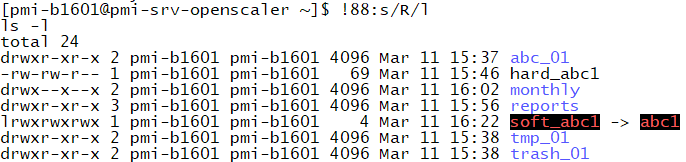
18. Получили информацию об оборудовании и операционной системе сервера. С помощью команды **cat /proc/cpuinfo** и команды **lscpu** получили сведения о процессоре. Первая команда содержит подробную информацию об отдельных ядрах CPU, а вторая команда выводит информацию об аппаратном обеспечении CPU в удобном для пользователя формате. С помощью команды **cat /proc/meminfo** получили сведения о параметрах памяти, а с помощью команды **uname** сведения о версии ОС, **uname –m** сведения об архитектуре, **uname –r** сведения о версии ядра



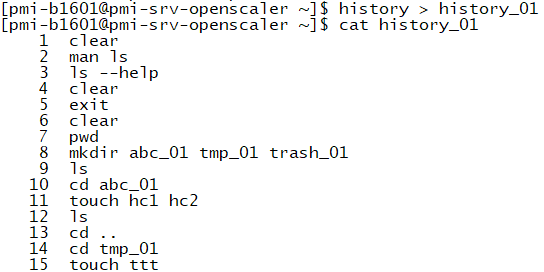
19 – 20. Создали жесткую и мягкую ссылки и проверили их работу



21. Выполнили модификацию команды из буфера команд



22. Сохранили историю команд в файле history\_01



1. Выводы по работе

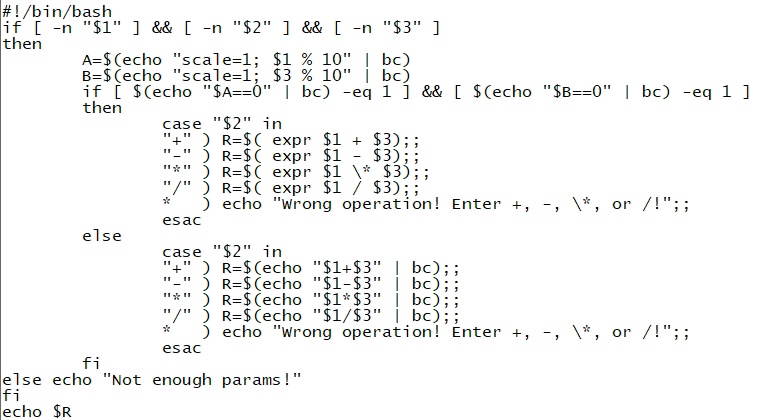
В ходе работы приобрели практические навыки работы с командным интерфейсом ОС Linux, а также проработали контрольные вопросы по теме.

# Лабораторная работа номер 2 Командные сценарии Linux

1. Цель работы:

изучение расширенных возможностей командного языка Shell.

1. Выполнение работы:
2. Посмотрели справку по команде **echo,** ознакомились с её ключами;
3. Освоили способы создания сценария, описанные в разделе 2.6;
4. Разработали сценарий, реализующий калькулятор с выполнением сложения, вычитания, умножения и деления;



1. Пример его запуска и результаты работы:

1) Работа сценария при некорректных входных данных



2) Работа сценария при отсутствии операции во входных данных



3) Работа сценария с делением вещественных чисел



4) Работа сценария с делением целых чисел



5) Работа сценария с умножением вещественных чисел



6) Работа сценария с умножением целых чисел



7)Работа сценария с вычитанием вещественных чисел



8)Работа сценария с вычитанием целых чисел



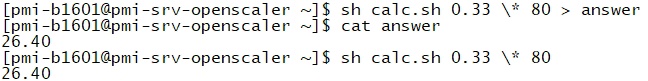
9) Работа сценария со сложением вещественных чисел



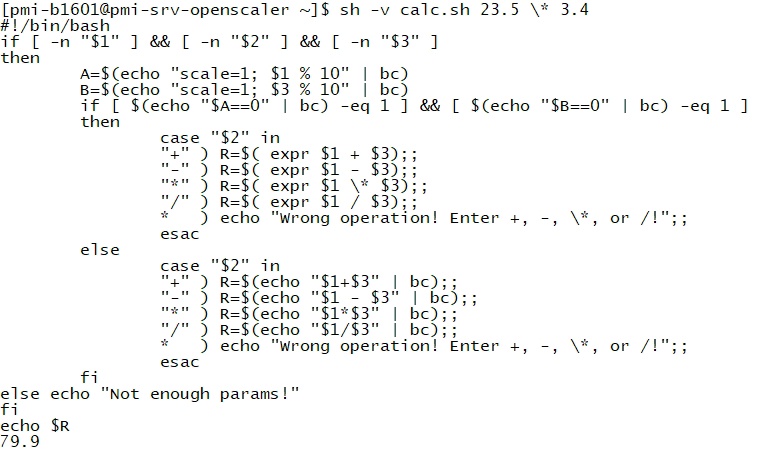
10) Работа сценария со сложением целых чисел



1. Вывели результат работы сценария в файл **answer**. Содержимое файла и результаты вывода на терминал одинаковы.



1. Запустили сценарий с ключами –v и –x. При запуске сценария с ключом –v выводилась каждая команда перед её выполнением без подстановки параметров, а при запуске с ключом –x выводилась каждая команда перед её выполнением с подстановкой параметров.



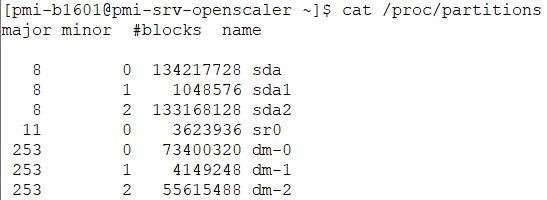
1. В ходе работы приобрели практические навыки работы с командными сценариями ОС Linux, а также проработали контрольные вопросы по теме.

# Лабораторная работа номер 4 Файловая система OC Linux

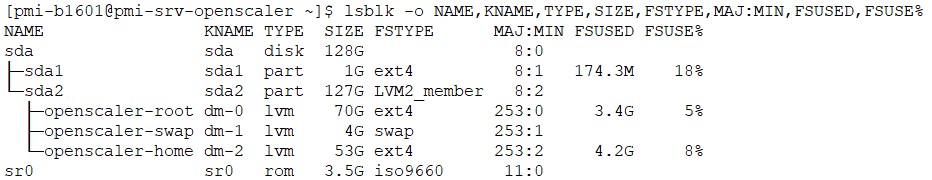
1. Цель работы:

изучение файловой системы ОС Linux и приобретение практических навыков применения команд для анализа файловой системы, управления файлами и процессами.

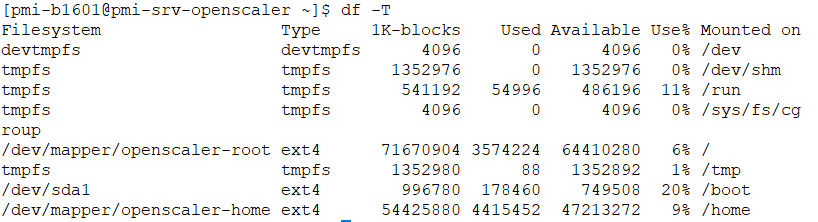
1. Выполнение работы:  
   1 Этап
2. Подключились к серверу ФПМИ, просмотрели файл **/proc/partitions**

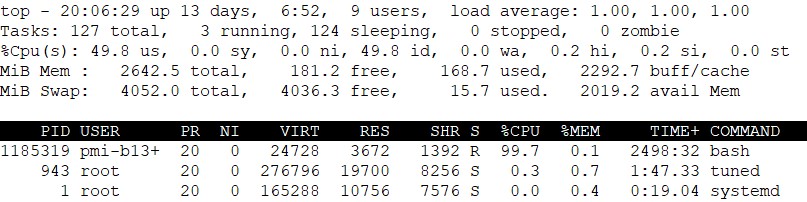


1. С помощью команд **lsblk** и **df** определили основные характеристики разделов внешней памяти сервера

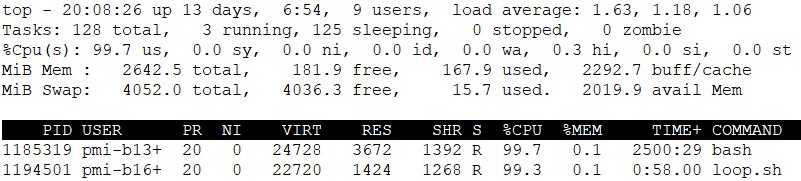
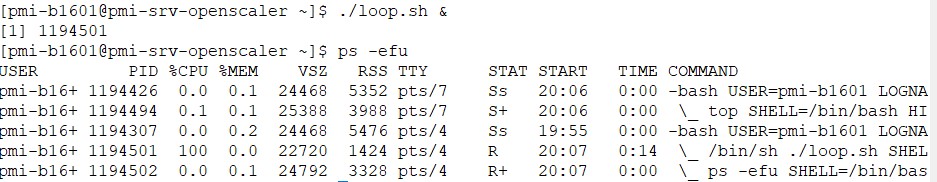


|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Имя устройства | Имя раздела | Тип раздела | Размер раздела | Тип ФС | Номер драйвера устройства | Коэф-т использования |
| 1 | sda | sda1 | первичный | 1G | ext4 | 8 | 18% |
| 2 | sda | sda2 | расширенный | 127G | LVM2\_ member | 8 |  |
| 3 | sda | dm-0 | логический | 70G | ext4 | 253 | 5% |
| 4 | sda | dm-1 | логический | 4G | swap | 253 |  |
| 5 | sda | dm-2 | логический | 53G | exp4 | 253 | 8% |
| 6 | sr0 | sr0 | первичный | 3.5G | iso9660 | 11 |  |

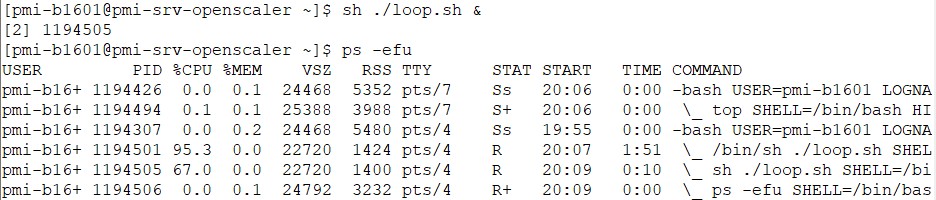
1. С помощью команд **df** и **du** определили типы файловых систем, используемых на сервере, а также в каком из имеющихся разделов расположен ваш домашний каталог и размер домашнего каталога.
2. Открыли второе соединение с сервером, в котором командой **top** включили мониторинг наших процессов



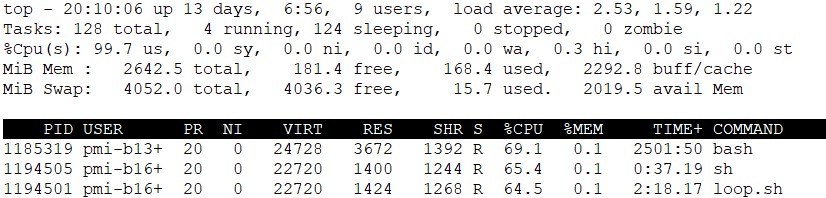
1. Создали в файле **loop.sh**сценарий, реализующий бесконечный цикл и запустили его в фоновом режиме командой **./loop.sh &**
2. С помощью команды **ps** **-efu** посмотрели список наших активных процессов. Посмотрели изменения в результатах, выводимых командой **top** в втором окне.



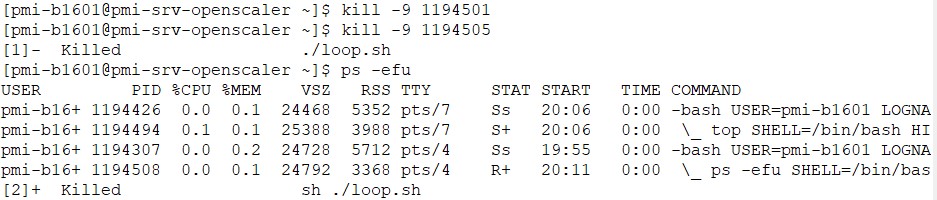
1. Повторно запустили сценарий командой **sh** **./loop.sh & ,** посмотрели список наших активных процессов



1. Посмотрели изменения в результатах, выводимых командой **top** в втором окне.

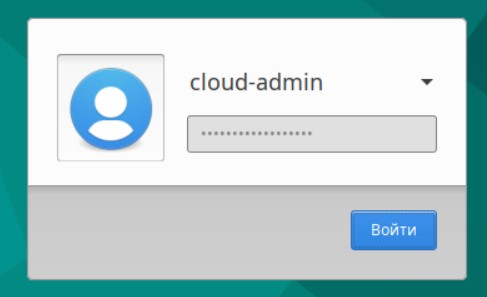
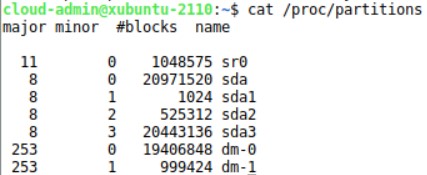
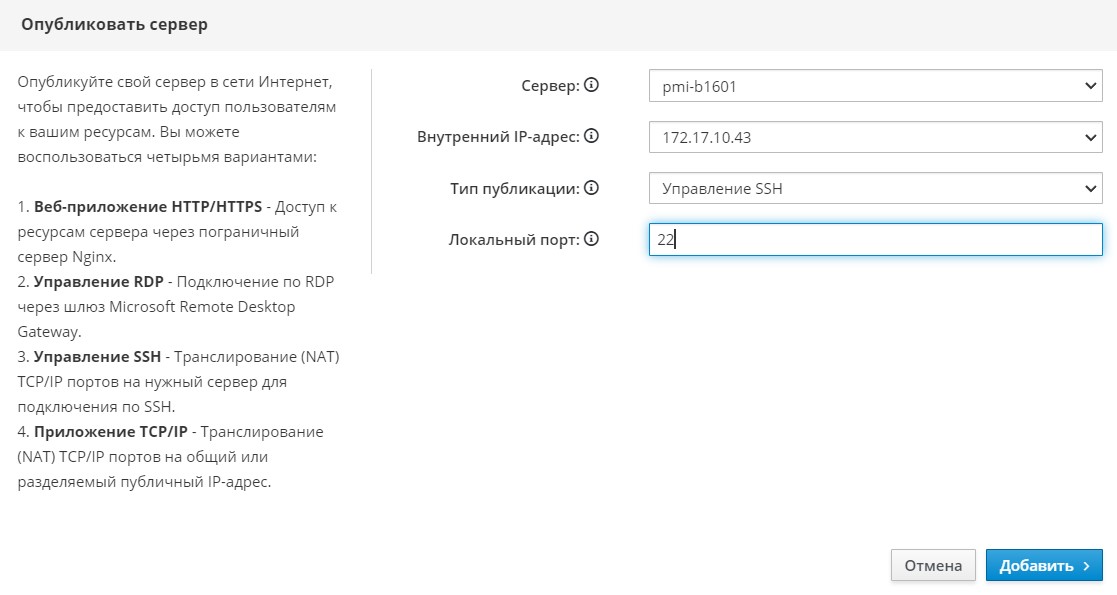
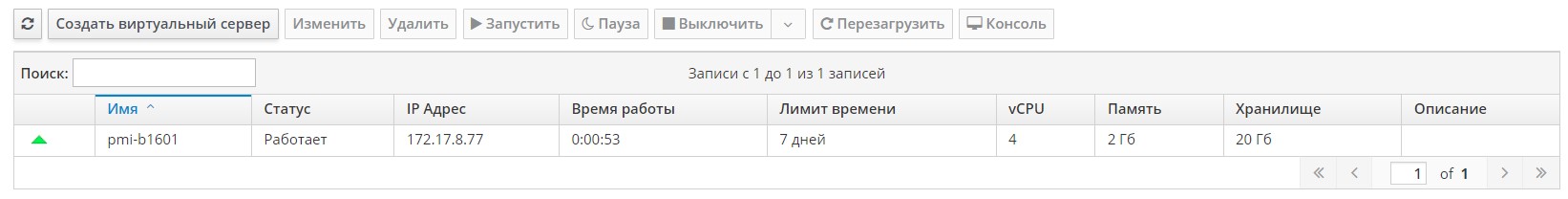
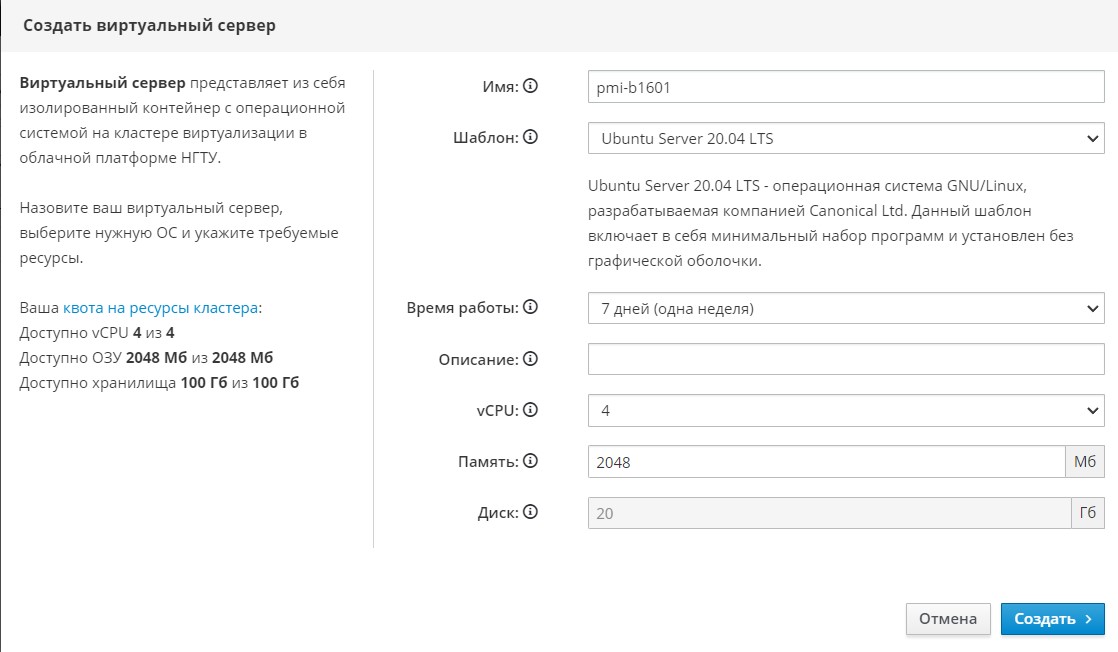


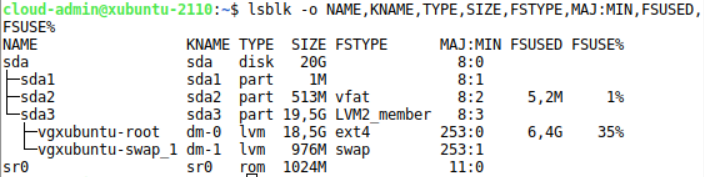
1. Выполнили принудительное прерывание всех процессов, запущенных в п.5 и п.7 и убедились, что все процессы уничтожены. Прервали во втором окне выполнение команды **top** и закрыли оба соединения.



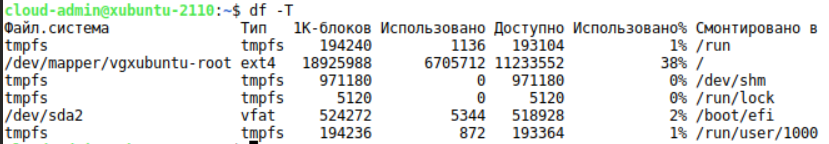
2 Этап

1. Создали на облачной платформе НГТУ виртуальный сервер (ВС) с операционной системой Ubuntu Server и опубликовали его для доступа из внешней сети по протоколу SSH.
2. 2. Подключились к ВС через консоль. Запустили эмулятор терминала, выполнили п.1 – п.3 первого этапа задания.

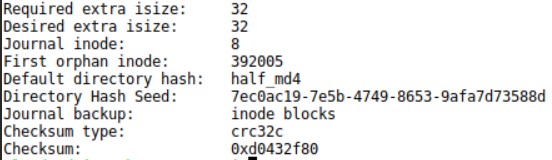




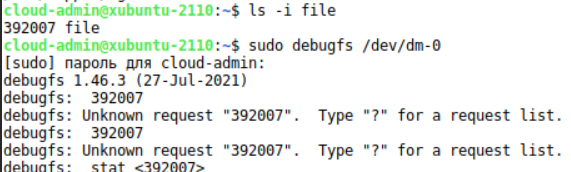
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № п/п | Имя устройства | Имя раздела | Тип раздела | Размер раздела | Тип ФС | Номер драйвера устройства | Коэф-т использования |
| 1 | sda | sda1 | первичный | 1M |  | 8 |  |
| 2 | sda | sda2 | первичный | 513M | vfst | 8 | 1% |
| 3 | sda | sda3 | расширенный | 19.5G | LVM2\_member | 8 |  |
| 4 | sda | dm-0 | логический | 18.5G | ext4 | 253 | 35% |
| 5 | sda | dm-1 | логический | 976M | swap | 253 |  |
| 6 | sr0 | sr0 | первичный | 1024M |  | 11 |  |

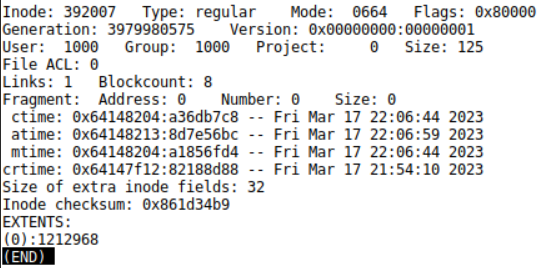


1. Просмотрели структуру суперблока файловой системы ВС с помощью команды **sudo tune2fs –l /dev/dm-0**.

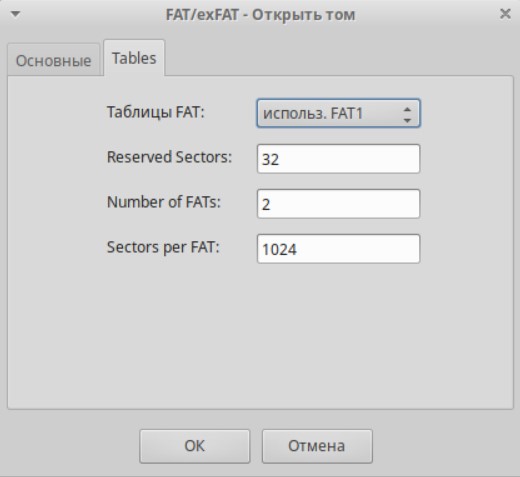
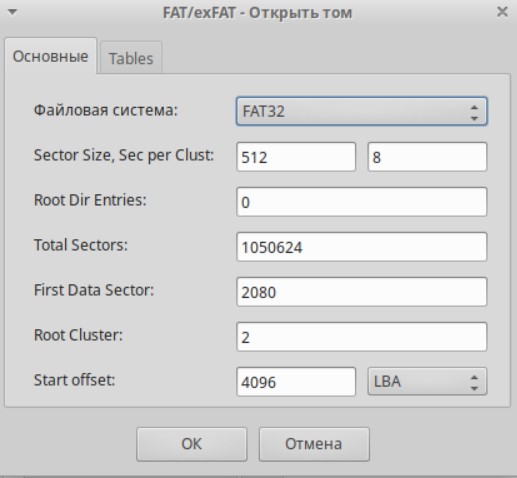
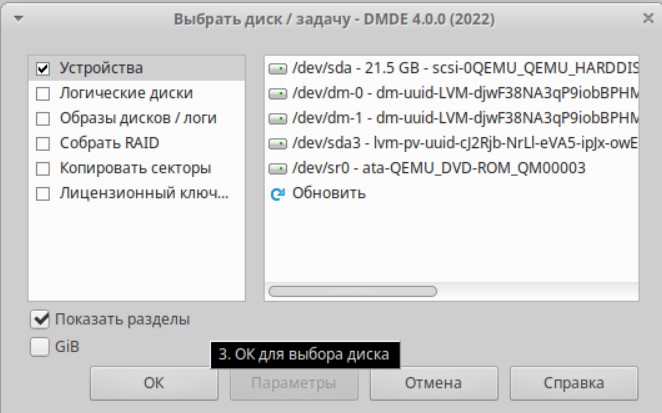


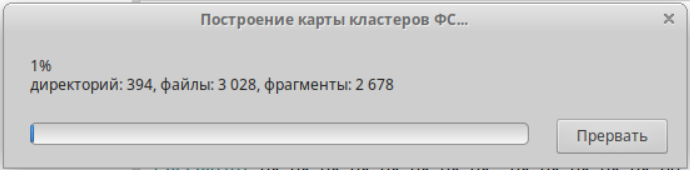
1. Создали в домашнем каталоге текстовый файл размером не менее 10 Кб, содержимое которого должно содержать строки из символов латиницы, включая фамилии членов бригады.
2. Определили номер дескриптора созданного файла командой **ls** и содержимое дескриптора командой **sudo debugfs /dev/dm-0**.

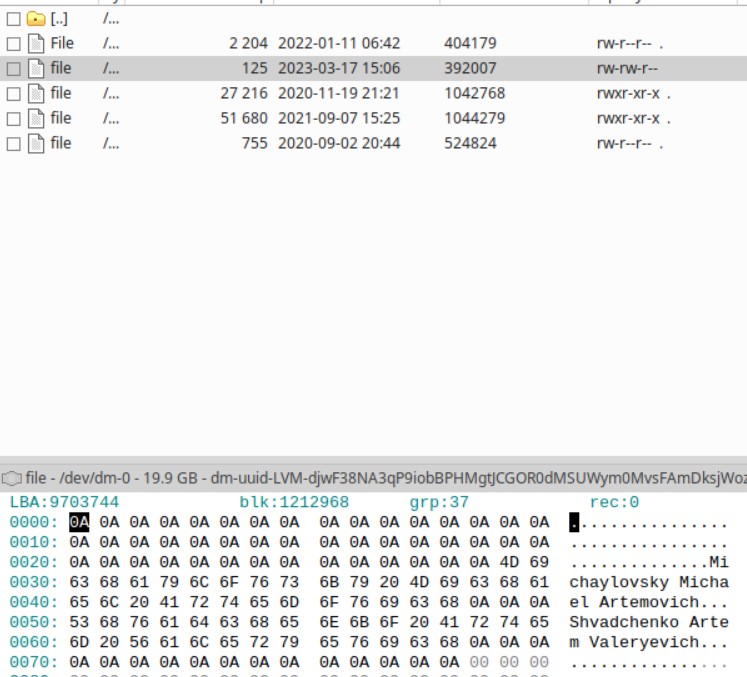
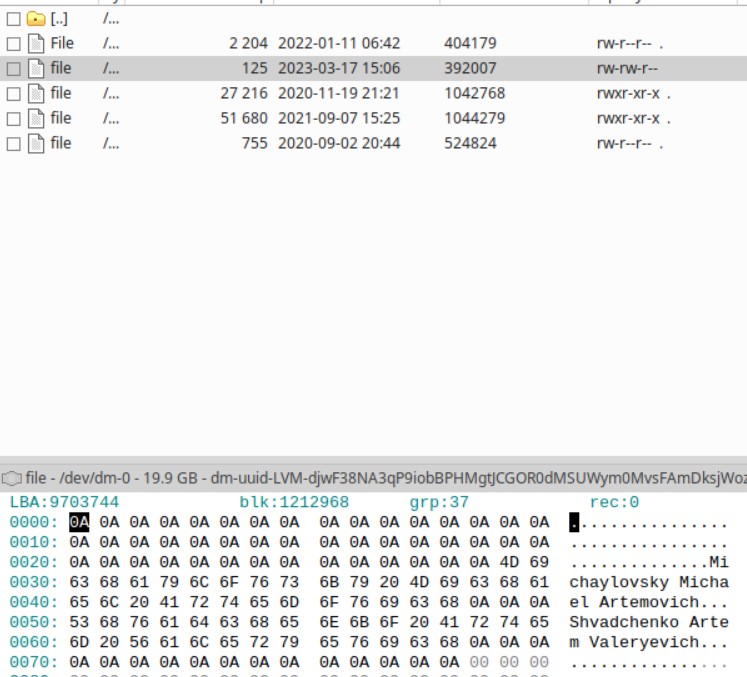


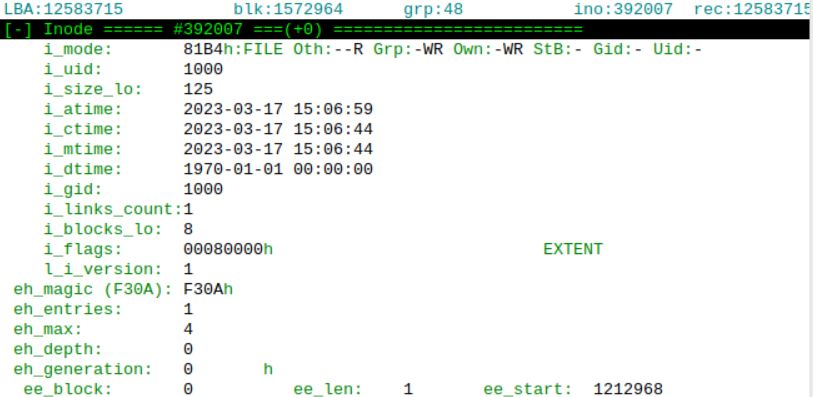


1. С помощью Applications Menu запустили программу Web Browser, cкачали в домашний каталог с сайта <https://dmde.ru/download.html> 64-битную версию дискового редактора DMDE для Linux и установили редактор.
2. Открыли устройство sda и определили его параметры. Открыли логический диск /dev/dm-0, просмотрели содержимое файла, созданного в п.4, и определили номер его дескриптора.

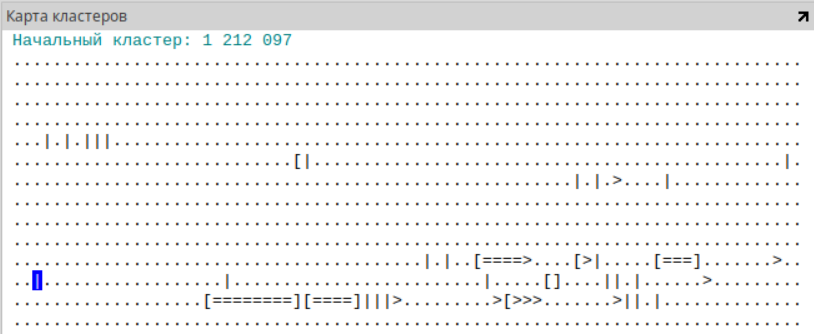


1. В меню Редактор/Файловая запись указали номер дескриптора, просмотрели его содержимое и нашли номера занимаемых файлом блоков





1. Построили карту кластеров и определили расположение файла на карте



1. Изучили файловой системы ОС Linux и приобрели практические навыки применения команд для анализа файловой системы, управления файлами и процессами. Проработали контрольные вопросы по теме.

# Лабораторная работа номер 5 Системы контроля версий

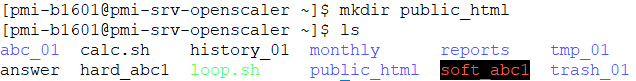
1. Цель работы:

Целью выполнения работы является изучение систем контроля версий git и GitLab, а также подготовка репозитория для выполнения лабораторной работы № 6.

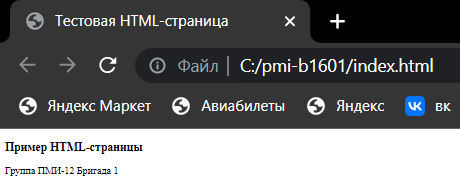
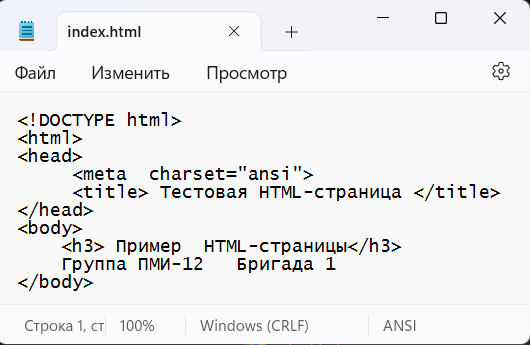
1. Выполнение работы:

Часть 1

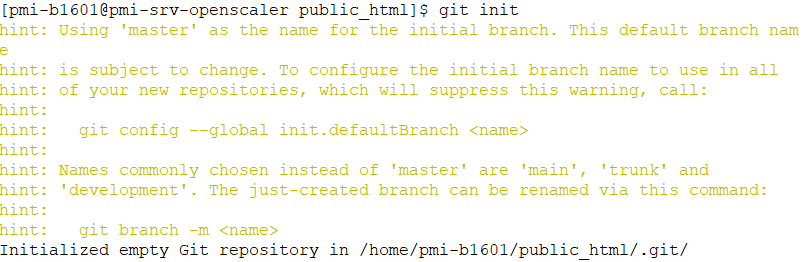
1. В домашнем каталоге Open Scaler создать каталог **public\_html**.



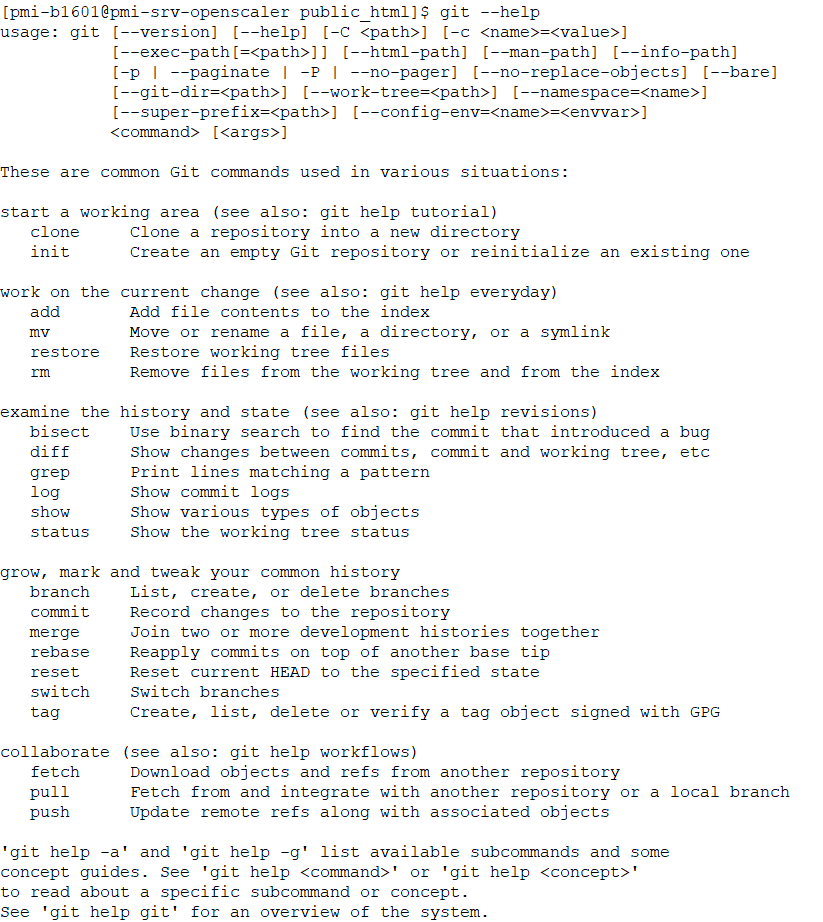
1. В Блокноте Windows создать текстовый файл в кодировке ANSI с именем **index.html**, текст которого приведен в разделе 2.4. Проверить работу этого файла и при отсутствии ошибок скопировать его в каталог **public\_html** сервера.



1. Перейти в каталог **public\_html** сервера и создать в нем локальный git-репозиторий.



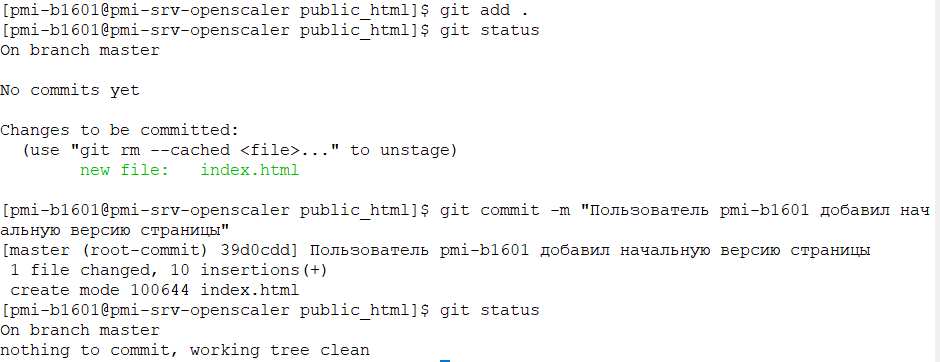
1. Вывести список команд git.



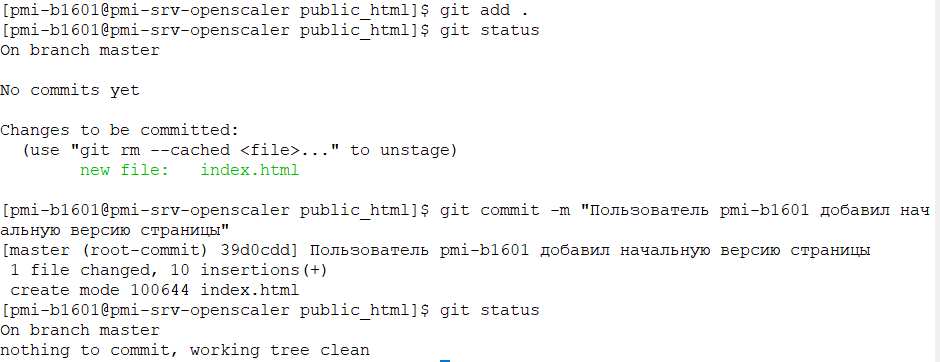
1. Выполнить глобальные настройки репозитория путем установки имени пользователя и адреса его электронной почты.



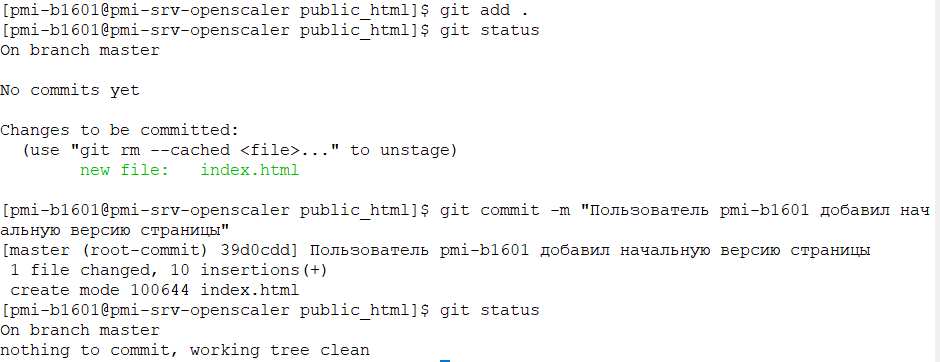
1. Добавить файл **index.html** в область индексирования репозитория.



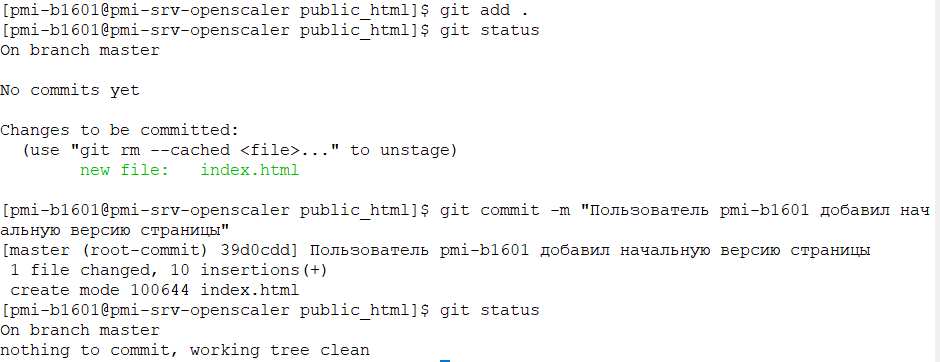
1. Определить состояние репозитория командой *status.*



1. Выполнить начальный коммит с добавлением комментария, например «Пользователь pmi-b1105 добавил начальную версию страницы»

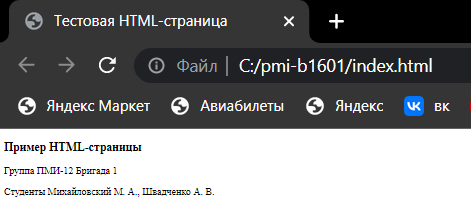
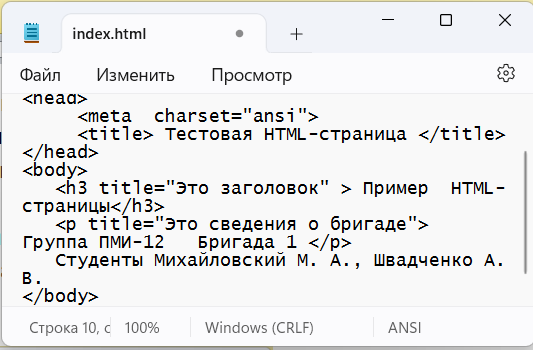


1. Повторно определить состояние репозитория, сравнить результаты с п. 7.

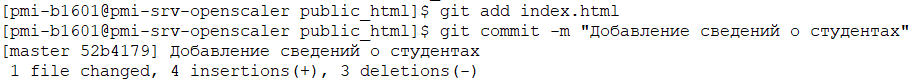


1. Выполнить все задания раздела 2.4 по редактированию файла **index.html.** Редактирование рекомендуетсяпроводить в среде Windows. После каждого внесения изменений необходимо после проверки корректности работы файла скопировать его в каталог **public\_html** сервера и сохранить обновленную версию в репозитории с соответствующим комментарием.

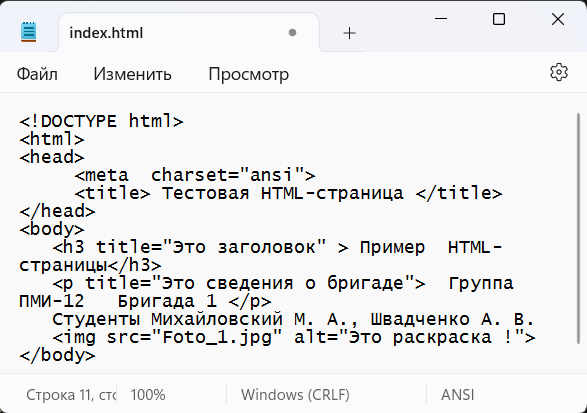
Вносим изменения в файл **index.html**: группу и номер бригады оформляем в виде абзаца, добавляем сведения о студентах бригады и всплывающие подсказки.

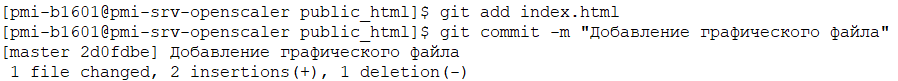


Добавляем измененный файл в индекс и выполняем коммит

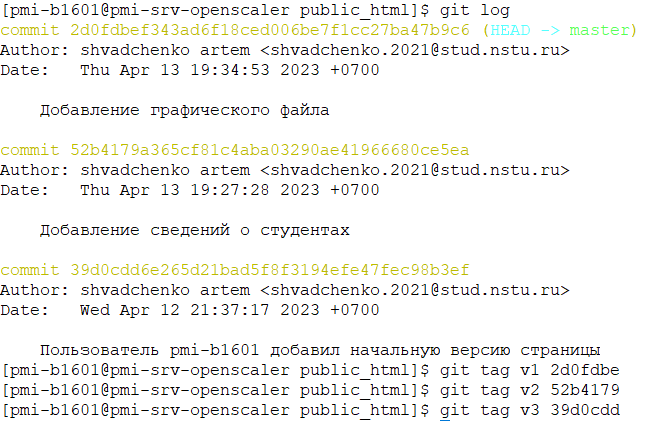


Изменяем файл **index.html (**добавляем любой графический файл) и выполняем коммит

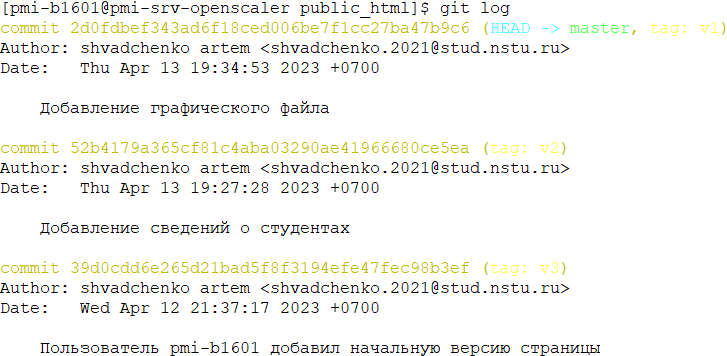




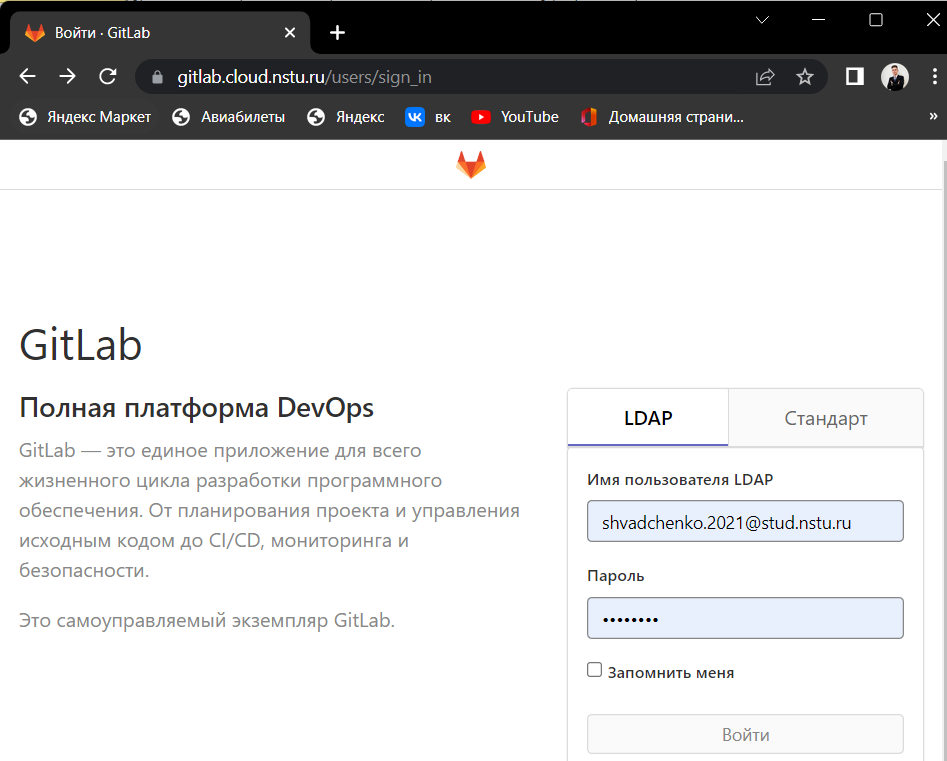
1. Создать отдельный тэг для каждого коммита.



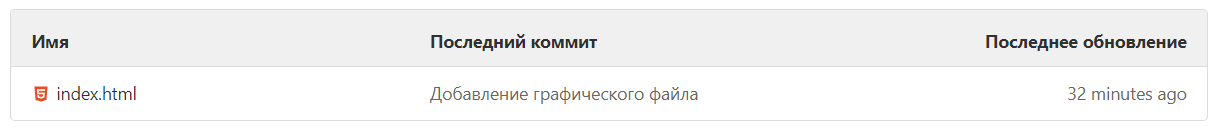
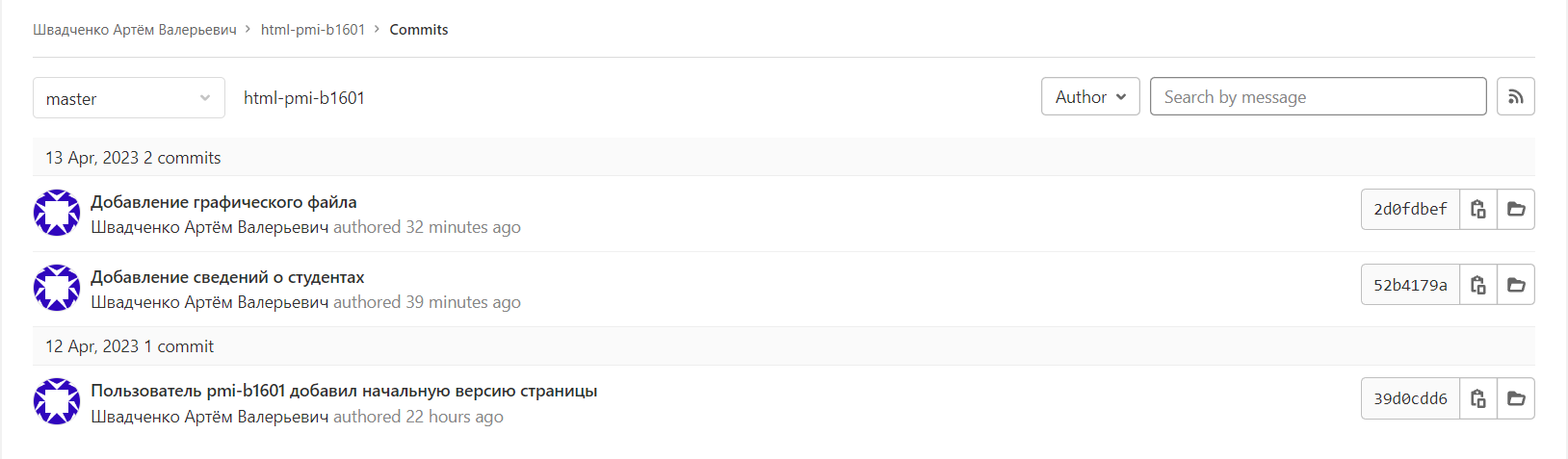
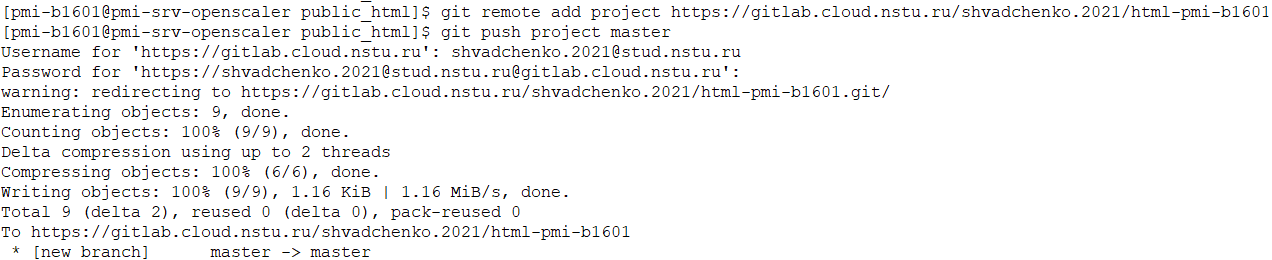
1. Вывести журнал всех изменений файла **index.html.**



1. Войти в систему GitLab на облачной платформе НГТУ путем ввода логина и пароля от личного кабинета.



1. Создать проект, заполнив соответствующую форму (см. рис.12). Имя проекта должно содержать номер Вашей группы и бригады, например, *html-pmi-b1105,* уровень видимости проекта – *internal.*
2. Загрузить все данные локального репозитория каталога **public\_html** в проект, созданный в п. 14.



Часть 2

1. В домашнем каталоге сервера Open Scaler создать рабочий каталог для разработки и отладки программ, его имя должно содержать бригадный логин, например, **git-pmi-b1105**.





1. Проверить работоспособность программ, разработанных по курсам «Структуры данных и алгоритмы» (для групп ПМ) или « Информационные технологии и основы программирования» (для групп ПМИ) в среде Windows. У каждого студента должен быть свой вариант программы (для бригады из двух человек далее – **prog1** и **prog2**).  
   prog1 – Швадченко А. В.

Дана непустая последовательность чисел (целых) используя структуры данных определить сумму последовательности

prog2 – Михайловский М. А.

Дана непустая последовательность чисел (целых) используя структуры данных определить среднее арифметическое последовательности

Содержание программы prog1.c:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

typedef struct sNode {

int value;

struct sNode\* next;

}list;

list\* read\_list1() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int u;

printf("Enter the test file number:");

scanf("%d", &u);

char\* file\_name = "";

if (u == 1)

{

file\_name = "elems1.txt\0";

}

else if (u == 2)

{

file\_name = "elems2.txt\0";

}

else if (u == 3) {

file\_name = "elems3.txt\0";

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

int val;

FILE\* file;

file = fopen(file\_name, "r");

while (fscanf(file, "%d", &val) != EOF) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = val;

p->next = NULL;

}

fclose(file);

return head;

}

list\* enter\_list1() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int n;

int value;

printf("Enter the number of totalized objects:");

scanf("%d", &n);

while (n > 0) {

if (scanf("%d", &value) == 1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = value;

p->next = NULL;

n--;

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

}

return head;

}

int sum\_list1(list\* head) {

int result = 0;

list\* p = head->next;

while (p != NULL) {

result += p->value;

p = p->next;

}

return result;

}

void summer1() {

int option;

printf("This program calculates the sum of the list items. the list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu\n");

printf("1 - Read data from file\n2 - Enter data on keyboard\nEnter a number for the operation:");

scanf("%d", &option);

list\* lst;

if (option == 1) {

lst = read\_list1();

if (lst->next == NULL) {

printf("Error! Invalid file number entered!\n");

}

}

else if (option == 2) {

lst = enter\_list1();

if (lst->next == NULL) {

printf("Error! An incorrect number of elements was entered or the element does not match the integer type!\n");

}

}

else {

printf("No such operation!\n");

lst = malloc(sizeof(list));

lst->next = NULL;

}

int sum = sum\_list1(lst);

printf("Total: %d", sum);

}

int main() {

summer1();

}

**Содержание программы prog2.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

typedef struct sNode {

int value;

struct sNode\* next;

}list;

list\* read\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int u;

printf("Enter the test file number:");

scanf("%d", &u);

char\* file\_name = "";

if (u == 1)

{

file\_name = "elems1.txt\0";

}

else if (u == 2)

{

file\_name = "elems2.txt\0";

}

else if (u == 3) {

file\_name = "elems3.txt\0";

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

int val;

FILE\* file;

file = fopen(file\_name, "r");

while (fscanf(file, "%d", &val) != EOF) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = val;

p->next = NULL;

}

fclose(file);

return head;

}

list\* enter\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int n;

int value;

printf("Enter the number of totalized objects:");

scanf("%d", &n);

while (n > 0) {

if (scanf("%d", &value) == 1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = value;

p->next = NULL;

n--;

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

}

return head;

}

float average\_list(list\* head) {

float sum = 0;

float count = 0;

list\* p = head->next;

if (p == NULL) {

return 0;

}

else {

while (p != NULL) {

sum += p->value;

count += 1;

p = p->next;

}

float result = sum / count;

return result;

}

}

int main() {

int option;

printf("This program calculates the arithmetic mean of the list items. The list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu\n");

printf("1 - Read data from file\n2 - Enter data on keyboard\nEnter a number for the operation:");

scanf("%d", &option);

list\* lst;

if (option == 1) {

lst = read\_list();

if (lst->next == NULL) {

printf("Error! Invalid file number entered!\n");

}

}

else if (option == 2) {

lst = enter\_list();

if (lst->next == NULL) {

printf("Error! An incorrect number of elements was entered or the element does not match the integer type!\n");

}

}

else {

printf("No such operation!\n");

lst = malloc(sizeof(list));

lst->next = NULL;

}

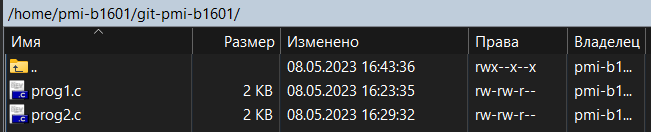
float avrg = average\_list(lst);

printf("Total: %f", avrg);

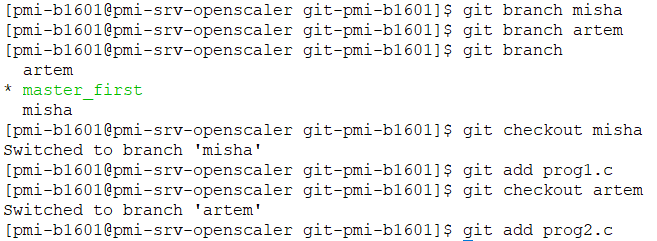
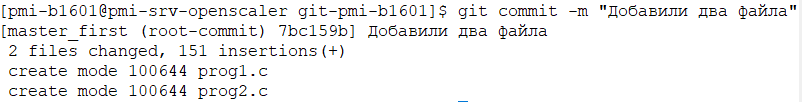
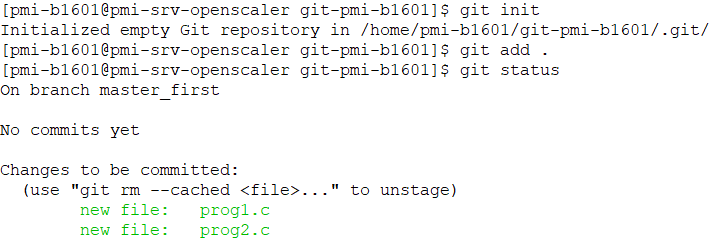
}

1. Скопировать файлы с исходными текстами программ в созданный рабочий каталог с помощью WinSCP.

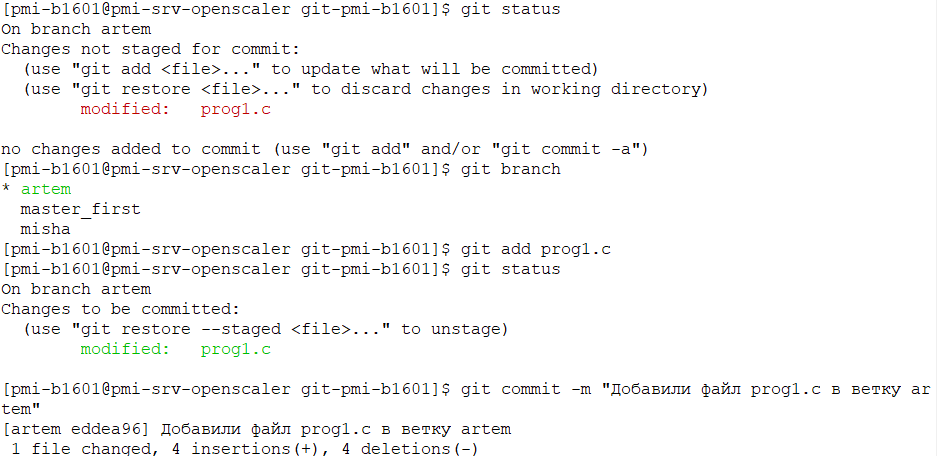




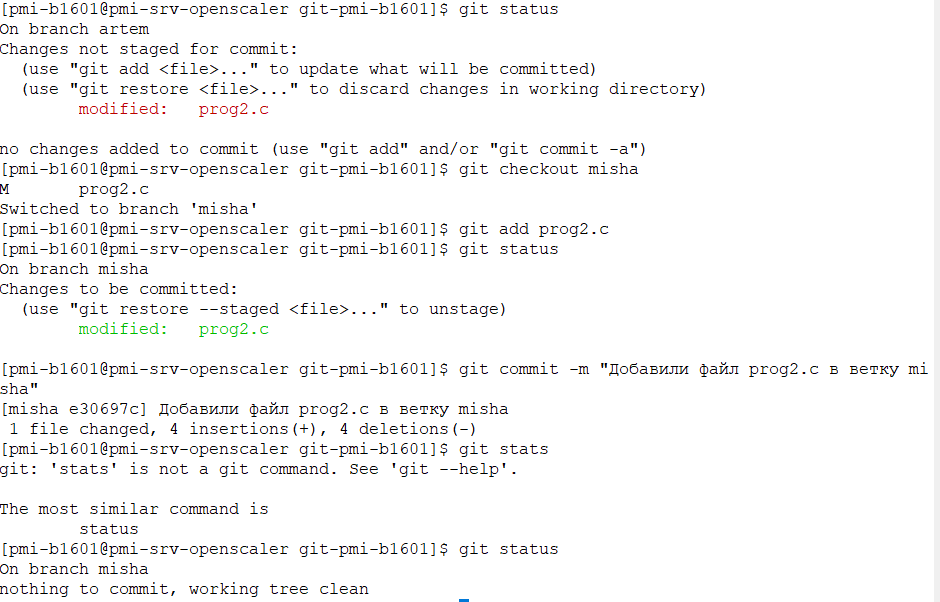
1. Перейти в рабочий каталог, создать в нем новый git-репозиторий и добавить в него исходные файлы проекта. Файлы **prog1.c** и **prog2.c** должны быть сохранены в разные ветки репозитория.











1. Доработать тексты программы с учетом требований





Проверка работы программы prog1

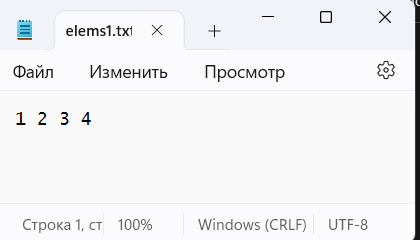


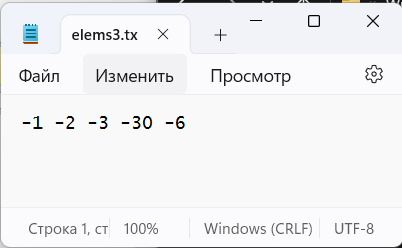
Проверка работы программы prog2



1. Разработать файлы с тестовыми наборами данных (не менее трех).

В тестовом файле elems1.txt содержатся целые положительные числа, в файле elems2.txt содержатся положительные и отрицательные числа, в файле elems3.txt содержатся отрицательные числа.



1. Выполнить слияние веток репозитория и разработать новую программу **mainprog**, главная функция которой формирует меню, состоящее из трех пунктов: вызов **prog1,** вызов **prog2** и завершение программы. Для этого код каждой из программ надо оформить в виде отдельной функции.



Текст программы mainprog.c:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

typedef struct sNode {

int value;

struct sNode\* next;

}list;

list\* read\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int u;

printf("Enter the test file number:");

scanf("%d", &u);

char\* file\_name = "";

if (u == 1)

{

file\_name = "elems1.txt\0";

}

else if (u == 2)

{

file\_name = "elems2.txt\0";

}

else if (u == 3) {

file\_name = "elems3.txt\0";

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

int val;

FILE\* file;

file = fopen(file\_name, "r");

int scan\_r = 1;

while (scan\_r != EOF) {

scan\_r = fscanf(file, "%d", &val);

if (scan\_r == 0) {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

else if (scan\_r != -1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = val;

p->next = NULL;

}

}

fclose(file);

return head;

}

list\* enter\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int n;

int value;

printf("Enter the number of totalized objects:");

scanf("%d", &n);

while (n > 0) {

if (scanf("%d", &value) == 1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = value;

p->next = NULL;

n--;

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

}

return head;

}

int sum\_list(list\* head) {

int result = 0;

list\* p = head->next;

while (p != NULL) {

result += p->value;

p = p->next;

}

return result;

}

float average\_list(list\* head) {

float sum = 0;

float count = 0;

list\* p = head->next;

if (p == NULL) {

return 0;

}

else {

while (p != NULL) {

sum += p->value;

count += 1;

p = p->next;

}

float result = sum / count;

return result;

}

}

int sum() {

int option;

// printf("\nThis program calculates the sum of the list items. the list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\n");

printf("1 - Read data from file\n2 - Enter data on keyboard\nEnter a number for the operation:");

scanf("%d", &option);

list\* lst;

if (option == 1) {

lst = read\_list();

if (lst->next == NULL) {

printf("Error! Invalid file number entered or the element does not match the integer type!\n");

}

}

else if (option == 2) {

lst = enter\_list();

if (lst->next == NULL) {

printf("Error! An incorrect number of elements was entered or the element does not match the integer type!\n");

}

}

else {

printf("No such operation!\n");

lst = malloc(sizeof(list));

lst->next = NULL;

}

int sum = sum\_list(lst);

return sum;

}

float average() {

int option;

// printf("\nThis program calculates the sum of the list items. the list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\n");

printf("1 - Read data from file\n2 - Enter data on keyboard\nEnter a number for the operation:");

scanf("%d", &option);

list\* lst;

if (option == 1) {

lst = read\_list();

if (lst->next == NULL) {

printf("Error! Invalid file number entered!\n");

}

}

else if (option == 2) {

lst = enter\_list();

if (lst->next == NULL) {

printf("Error! An incorrect number of elements was entered or the element does not match the integer type!\n");

}

}

else {

printf("No such operation!\n");

lst = malloc(sizeof(list));

lst->next = NULL;

}

float avrg = average\_list(lst);

return avrg;

}

int main() {

int option;

int result1;

float result2;

// printf("The prog1 program calculates the sum of the list items. \n");

//printf("The prog2 program calculates the arithmetic mean of the list items.\nThe list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\n");

//printf("\n1 - prog1\n2 - Eny key - exit\nEnter a number for the program:");

printf("1 - The first program calculates the sum of the list items.\n2 - The second program calculates the arithmetic mean of the list items.\nThe list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\nEnter a number for the program:");

scanf\_s("%d", &option);

if (option == 1) {

result1 = sum();

printf("Result of prog1: %i\n", result1);

}

else if (option == 2) {

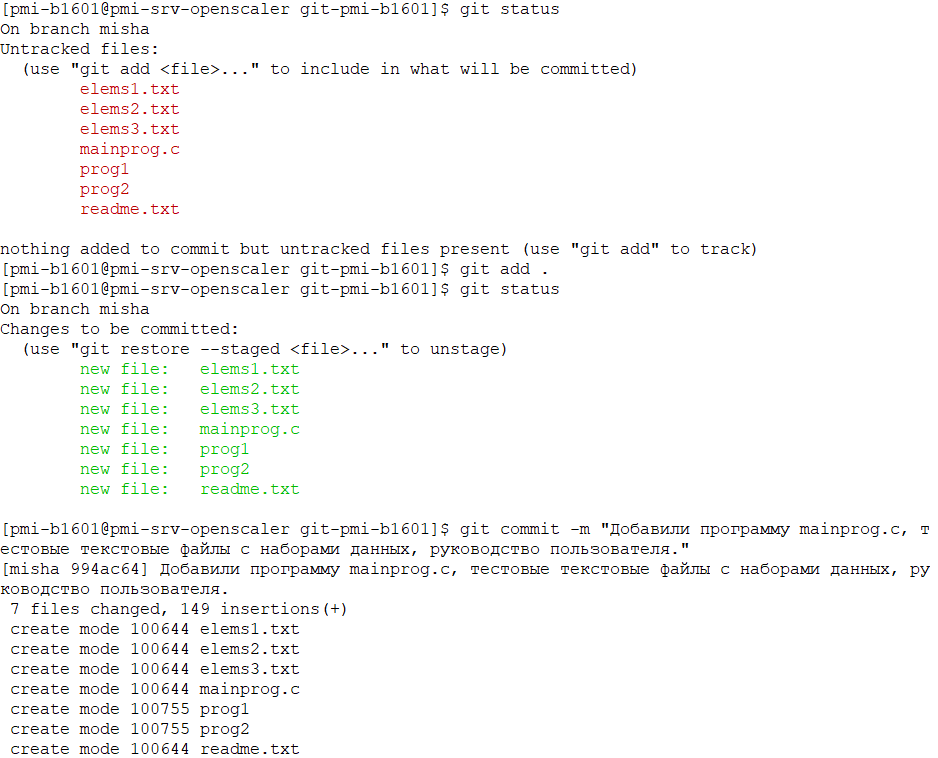
result2 = average();

printf("Result of prog2: %f\n", result2);

}

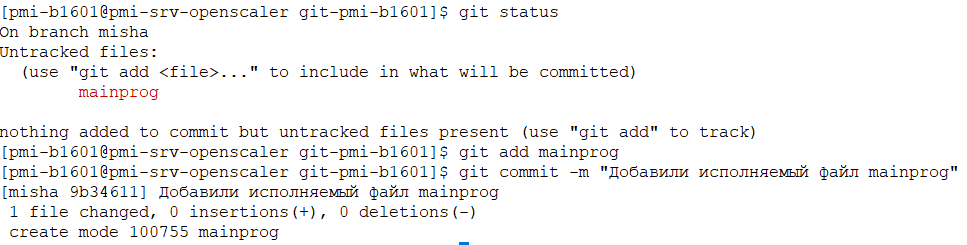
}

1. Сохранить исходные и исполняемые файлы всех программы,а также файлы с руководством пользователя и тестовыми наборами данных в одной из веток репозитория.



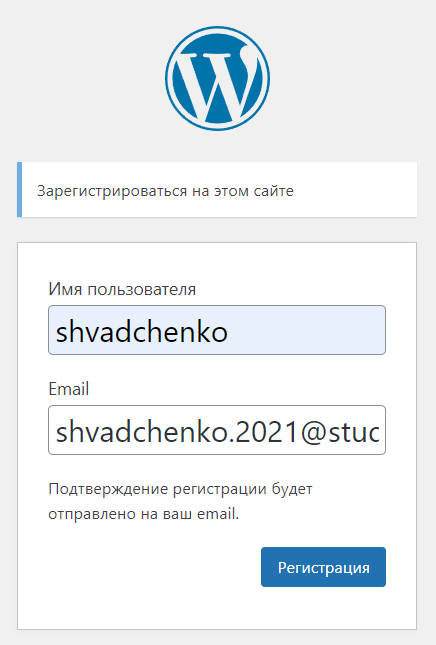
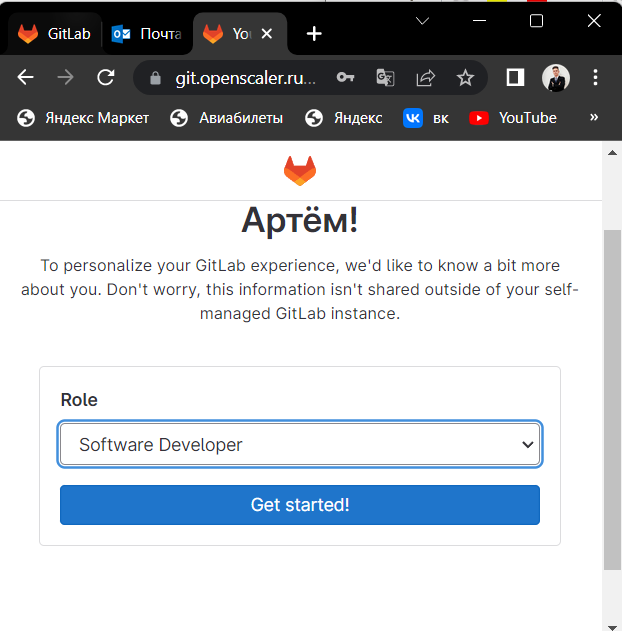
Формирование исполняемого файла mainprog



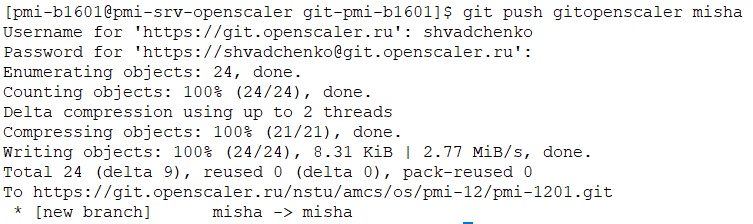


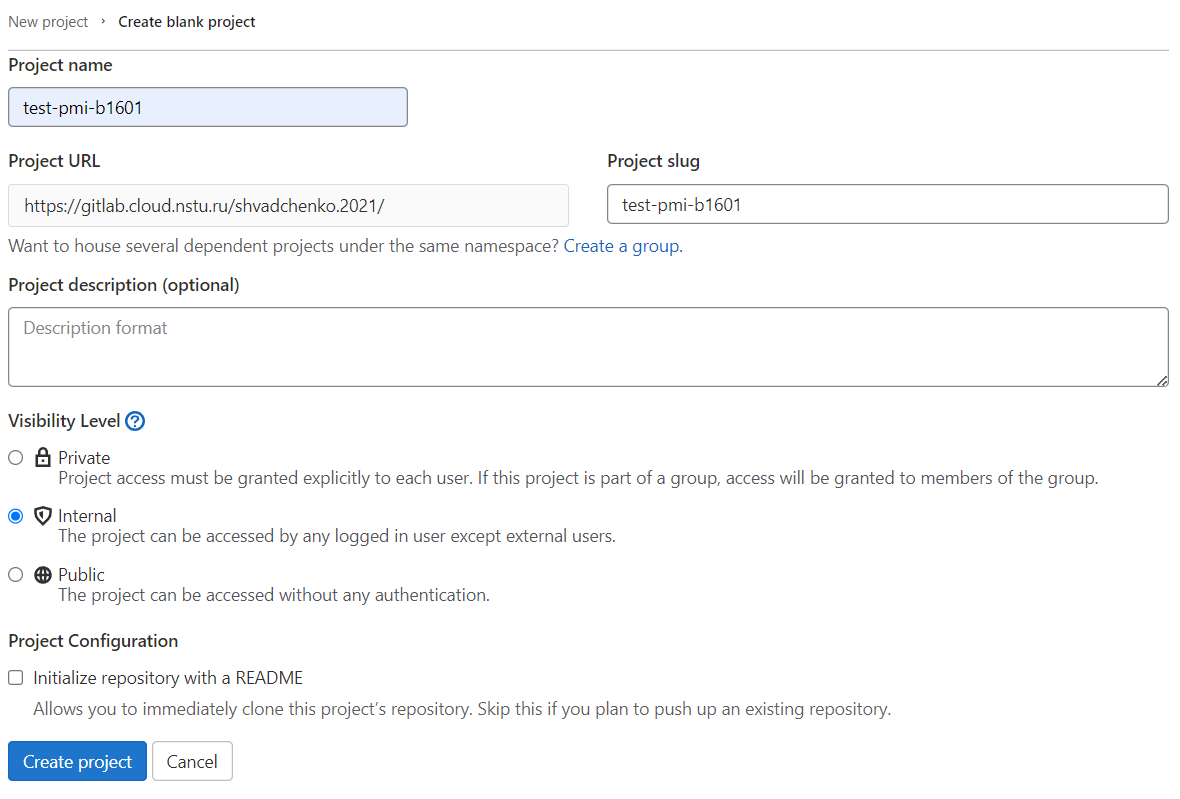
1. Зарегистрироваться в центральном репозитории сообщества *git.openscaler.ru* и загрузить в него все исходные и исполняемые файлы из локального репозитория. Имя проекта уточнить у преподавателя. При работе в системе GitLab облачной платформы НГТУ создать новый проект *test-pmi-b1105* и загрузить в него все файлы из локального репозитория.

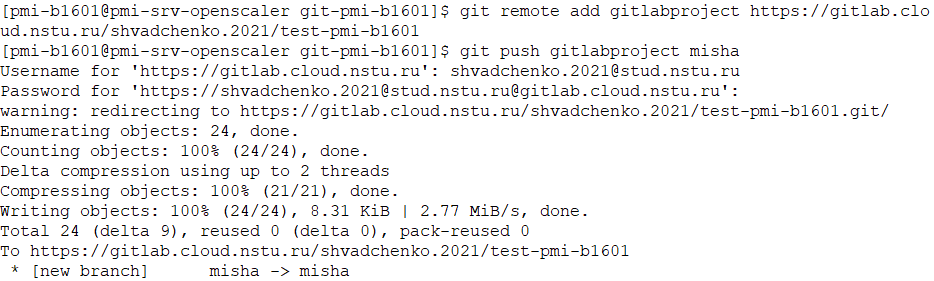
Регистрация на git.openscaler.ru и загрузка всех файлов в проект pmi-1201.

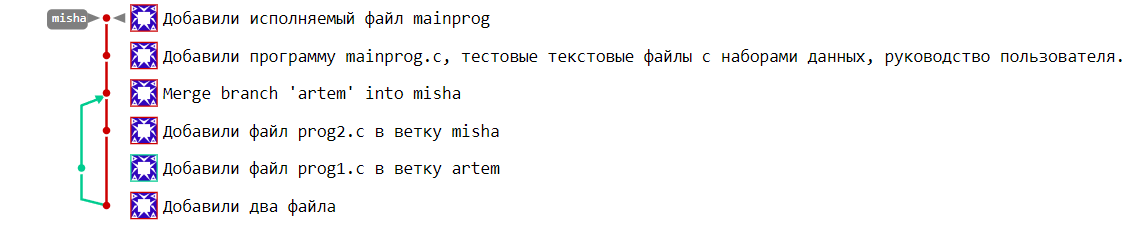






Создание проекта на GitLab



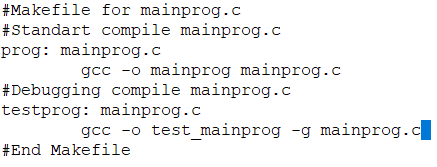


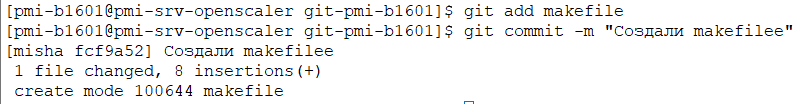
1. Изучили системы контроля версий. Проработали контрольные вопросы по теме.

# Лабораторная работа номер 6 Разработка программ в среде Linux

1. Цель работы:
2. Целью работы является изучение основных этапов разработки и отладки приложений в ОС Linux, а также приобретение практических навыков по использованию инструментальных средств фонда свободного программного обеспечения при компиляции исходного кода, сборке, отладке и тестировании программ, написанных на языке С.
3. Выполнение работы:
4. С помощью редактора **vi** создайте в рабочем каталоге make-файл, указав в качестве аргументов команд имена ваших файлов (см. п. 2.3). Основные сведения по редактору приведены в приложении 1.



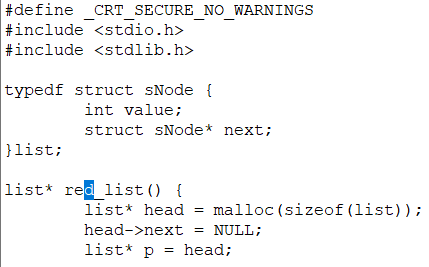
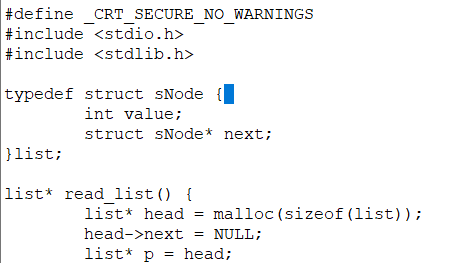




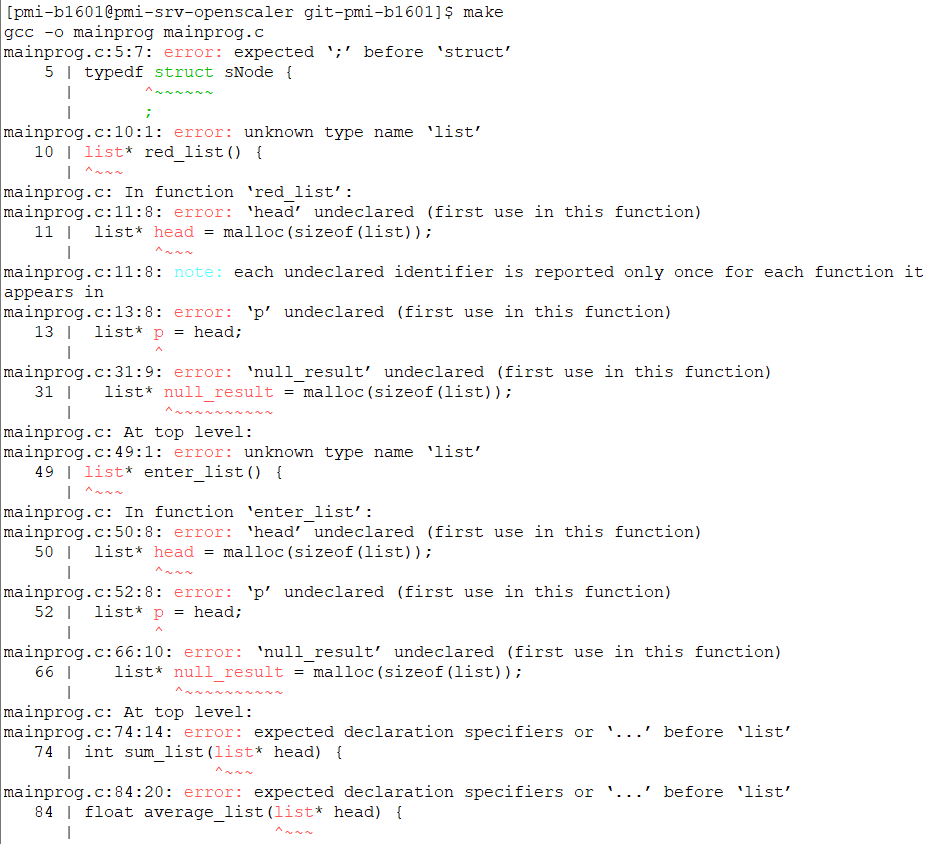
1. Выполните компиляцию файла **mainprog.c** с помощью make-файла, используя правило ***prog***. Если при компиляции были обнаружены ошибки, то используйте для их исправления редактор vi. Каждую исправленную версию программы сохраните в репозитории с поясняющим комментарием. Сведения об ошибках занесите в отчет (номер строки, значение строки до устранения и после устранения ошибки, пояснения).

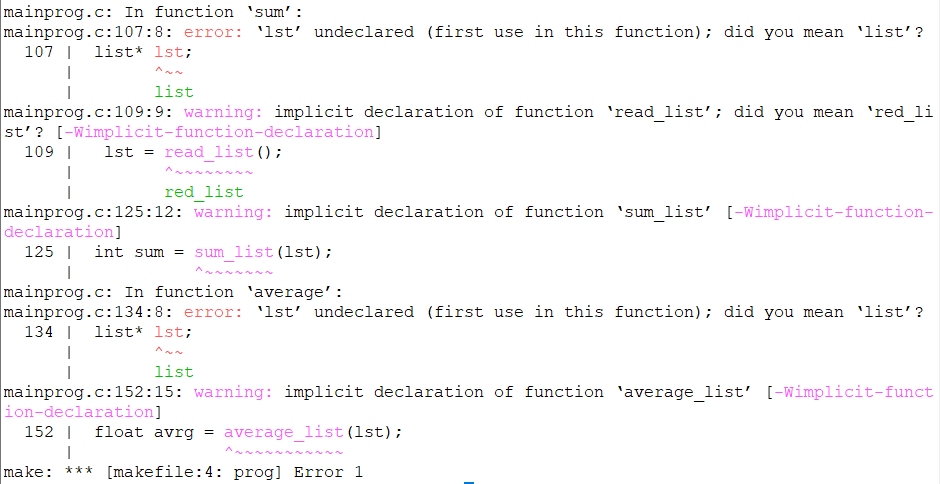
Если ошибок не обнаружено, то внесите в текст программы **mainprog** не менее двух синтаксических ошибок и повторно выполните компиляцию.



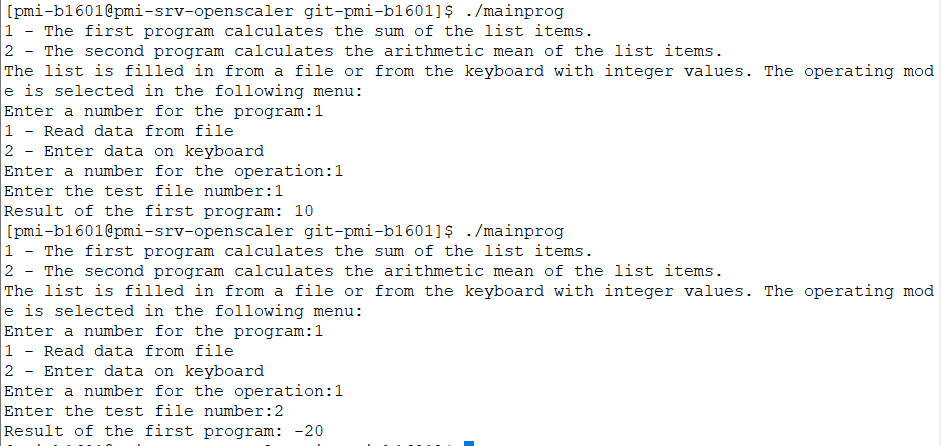


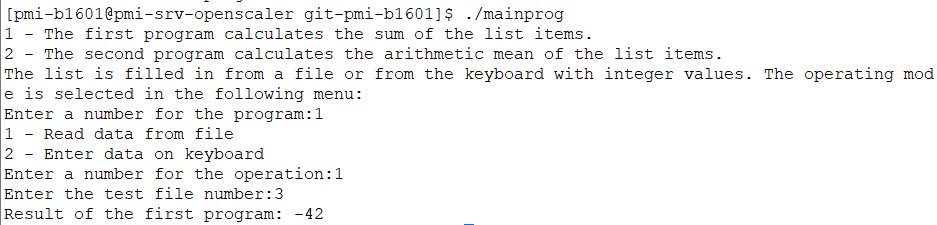
Добавили две ошибки: 5 строка вместо typedef написали typedf, 10 строка вместо read\_list написали red\_list.

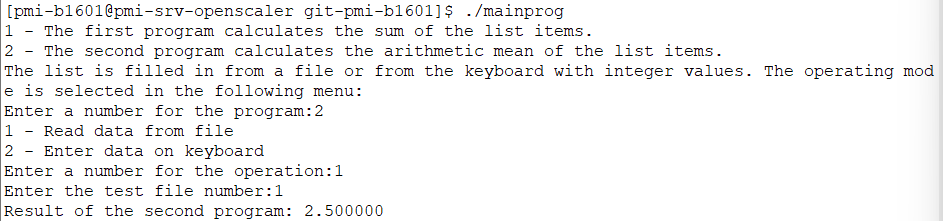


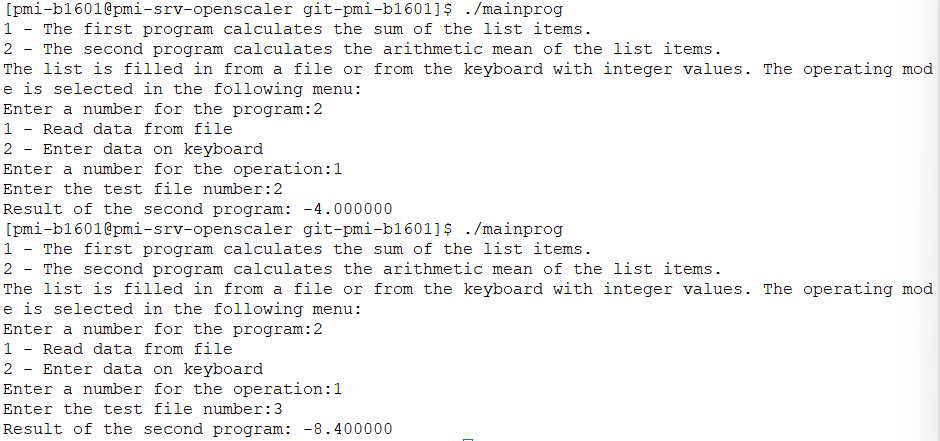


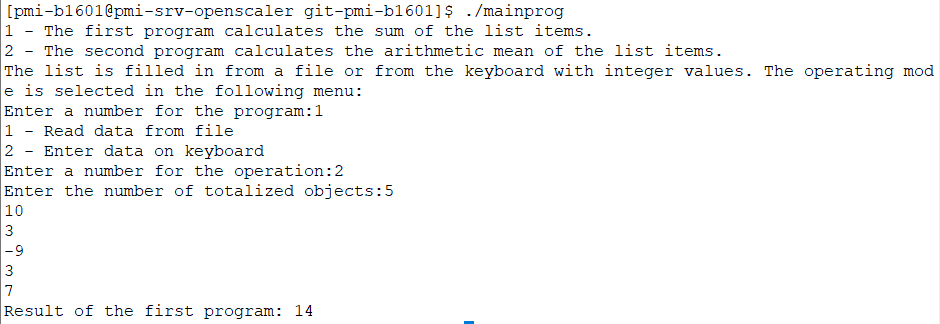
1. Запустите исполняемый файл **mainprog** и проверьте программу на подготовленных наборах тестовых данных, результаты тестирования занесите в отчет.

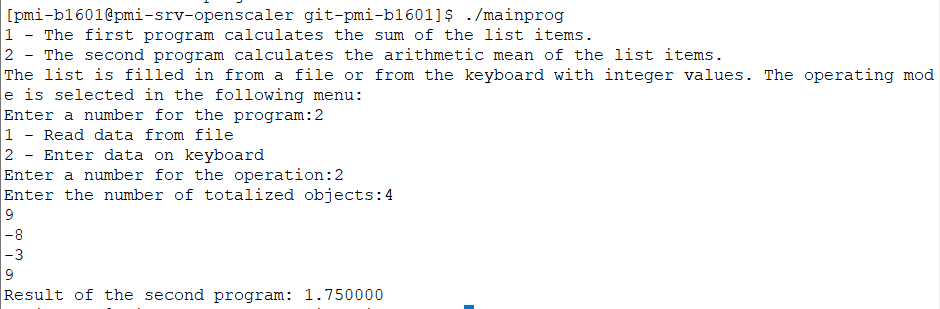








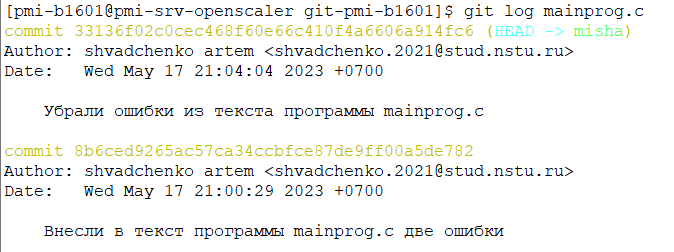




1. При обнаружении семантических ошибок перекомпилируйте программу с помощью правила ***testprog*** make-файла и запустите отладчик **gdb**, с помощью которого найдите причины появления ошибок. Проверьте программу на всех наборах тестовых данных. Сведения об ошибках занесите в отчет, исправленную версию программы сохраните в репозитории с поясняющим комментарием.

Семантических ошибок не обнаружено.

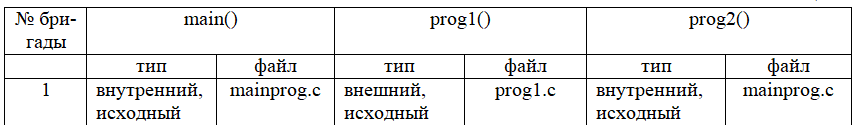
1. Выведите список всех изменений файла **mainprog.c**, выполненных в ходе отладки программы, занесите список в отчет.



1. Определите размер исполняемого модуля отлаженной программы. Удалите всю отладочную информацию и снова определите размер исполняемого модуля, сравните с предыдущим результатом, результат сравнения занесите в отчет.



1. Выполните разбиение программы **mainprog** на функции в соответствии с номером бригады из таблицы 2. Обратите внимание на тип функции (внутренняя или внешняя), тип файла (.c, .h или .o) и тип модуля (исходный или объектный). Занесите в отчет измененный текст программы.



Изменённый код программы mainprog.c

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

extern int prog1();

typedef struct sNode {

int value;

struct sNode\* next;

}list;

list\* read\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int u;

printf("Enter the test file number:");

scanf("%d", &u);

char\* file\_name = "";

if (u == 1)

{

file\_name = "elems1.txt\0";

}

else if (u == 2)

{

file\_name = "elems2.txt\0";

}

else if (u == 3) {

file\_name = "elems3.txt\0";

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

int val;

FILE\* file;

file = fopen(file\_name, "r");

int scan\_r = 1;

while (scan\_r != EOF) {

scan\_r = fscanf(file, "%d", &val);

if (scan\_r == 0) {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

else if (scan\_r != -1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = val;

p->next = NULL;

}

}

fclose(file);

return head;

}

list\* enter\_list() {

list\* head = malloc(sizeof(list));

head->next = NULL;

list\* p = head;

int n;

int value;

printf("Enter the number of totalized objects:");

scanf("%d", &n);

while (n > 0) {

if (scanf("%d", &value) == 1) {

p->next = malloc(sizeof(list));

p = p->next;

p->value = value;

p->next = NULL;

n--;

}

else {

list\* null\_result = malloc(sizeof(list));

null\_result->next = NULL;

return null\_result;

}

}

return head;

}

int sum\_list(list\* head) {

int result = 0;

list\* p = head->next;

while (p != NULL) {

result += p->value;

p = p->next;

}

return result;

}

float average\_list(list\* head) {

float sum = 0;

float count = 0;

list\* p = head->next;

if (p == NULL) {

return 0;

}

else {

while (p != NULL) {

sum += p->value;

count += 1;

p = p->next;

}

float result = sum / count;

return result;

}

}

float average() {

int option;

// printf("\nThis program calculates the sum of the list items. the list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\n");

printf("1 - Read data from file\n2 - Enter data on keyboard\nEnter a number for the operation:");

scanf("%d", &option);

list\* lst;

if (option == 1) {

lst = read\_list();

if (lst->next == NULL) {

printf("Error! Invalid file number entered or the element does not match the integer type!\n");

}

}

else if (option == 2) {

lst = enter\_list();

if (lst->next == NULL) {

printf("Error! An incorrect number of elements was entered or the element does not match the integer type!\n");

}

}

else {

printf("No such operation!\n");

lst = malloc(sizeof(list));

lst->next = NULL;

}

float avrg = average\_list(lst);

return avrg;

}

int main() {

int option;

int result1;

float result2;

// printf("The prog1 program calculates the sum of the list items. \n");

//printf("The prog2 program calculates the arithmetic mean of the list items.\nThe list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\n");

// printf("Eny key - exit\n");

printf("1 - The first program calculates the sum of the list items.\n2 - The second program calculates the arithmetic mean of the list items.\n");

printf("Eny key - exit\n");

printf("The list is filled in from a file or from the keyboard with integer values. The operating mode is selected in the following menu:\nEnter a number for the program:");

scanf("%d", &option);

if (option == 1) {

result1 = prog1();

printf("Result of first program: %i\n", result1);

}

else if (option == 2) {

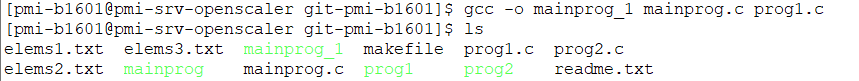
result2 = average();

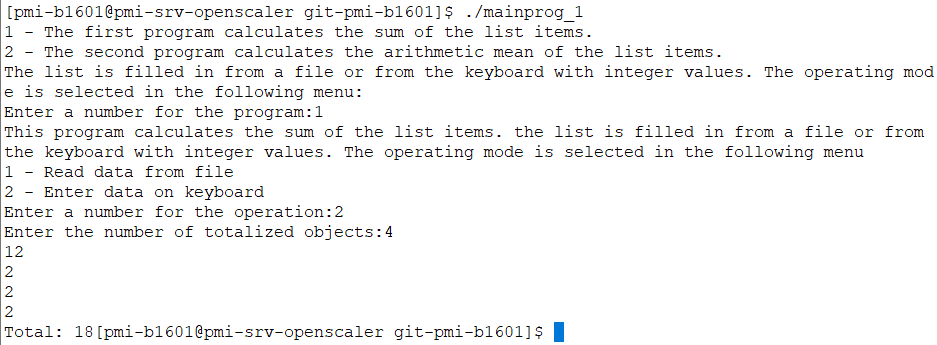
printf("Result of second program: %f\n", result2);

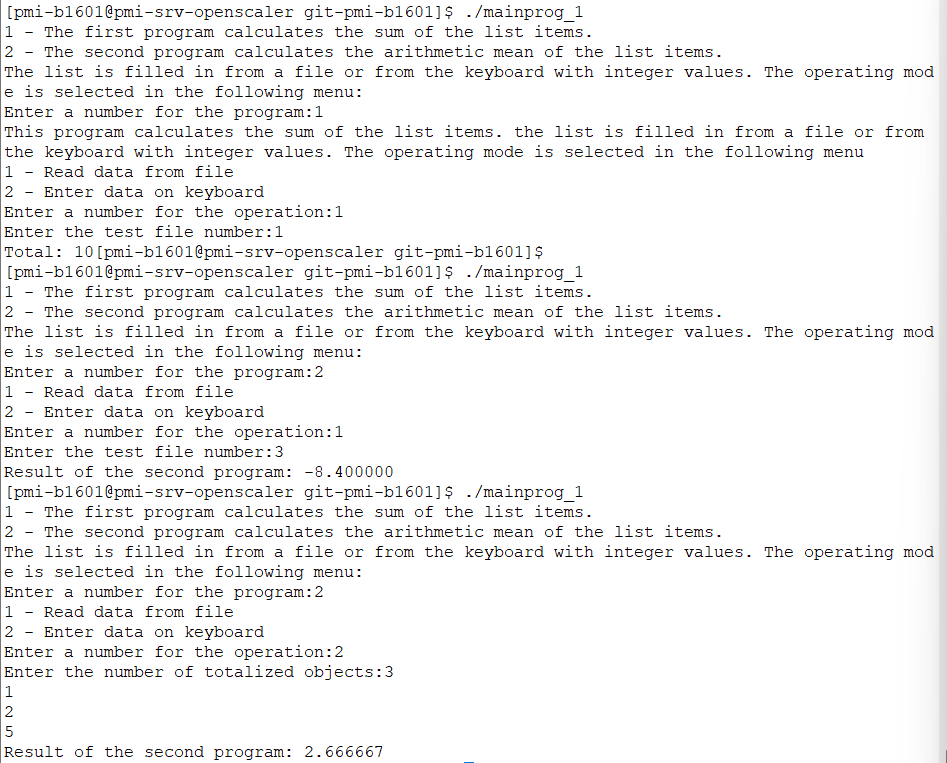
}

}

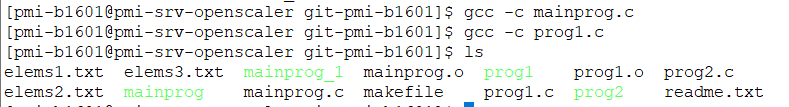
1. Выполните сборку программы в соответствии вариантом задания, используя неявный вызов компоновщика и задав имя исполняемого файла **mainprog\_1,** проверьте корректность работы программы и занесите в отчет результаты ее тестирования.

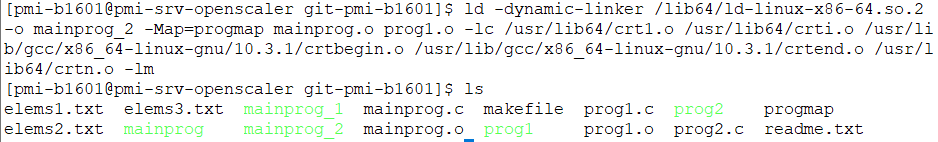


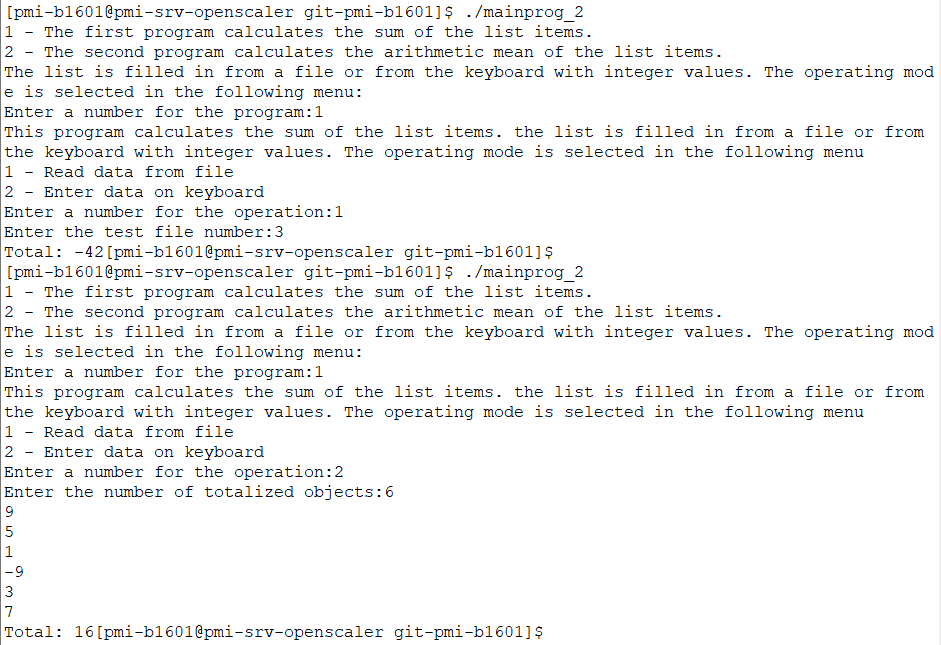


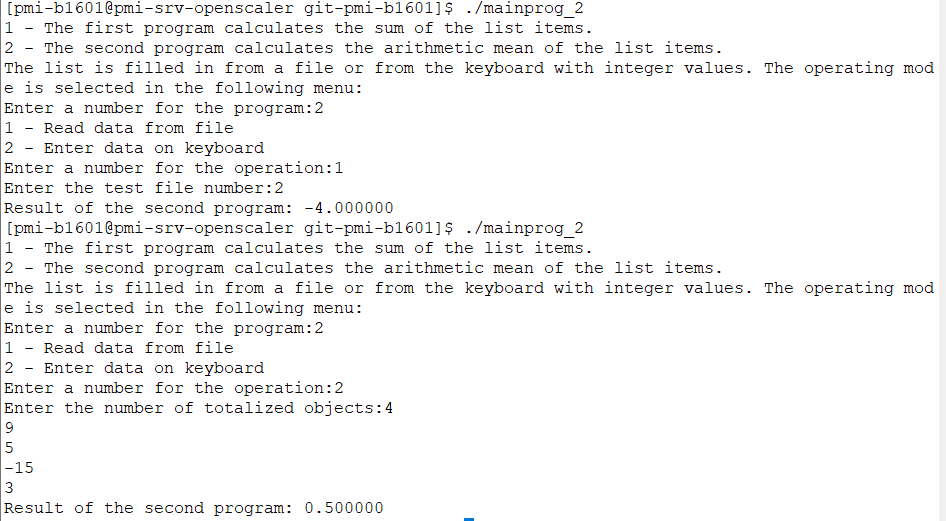


1. Выполните сборку программы в соответствии вариантом задания, используя явный вызов компоновщика. Результатом сборки должны быть исполняемый файл **mainprog\_2** и карта памяти **progmap;** проверьте корректность работы программы и занесите в отчет результаты ее тестирования.



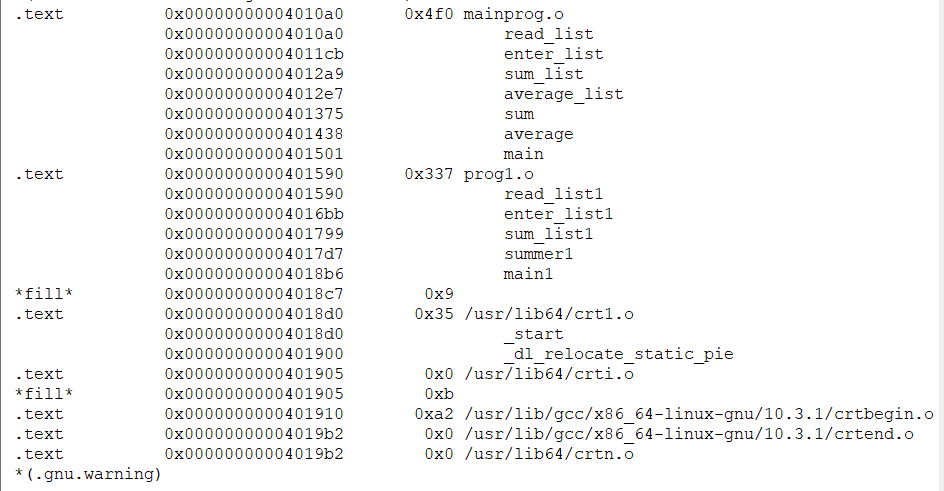


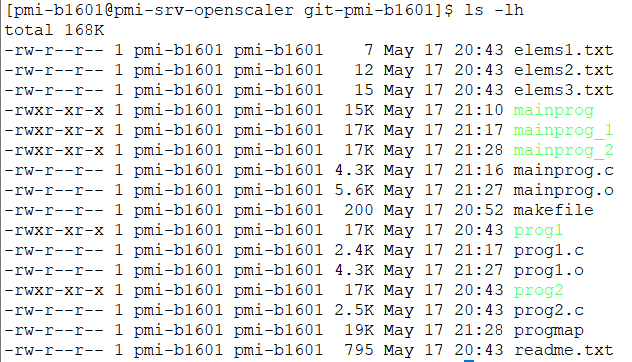




1. Из карты памяти **progmap** определите размеры машинного кода модулей **mainprog.o,** **prog1.o** и **prog2.o,** сравните их с размерами исходного и объектного кода этих модулей. Размеры файлов типа **.с** и **.о** определите с помощью команды ls. Результат представьте в виде таблицы 3, все данные должны быть подтверждены скриншотами.

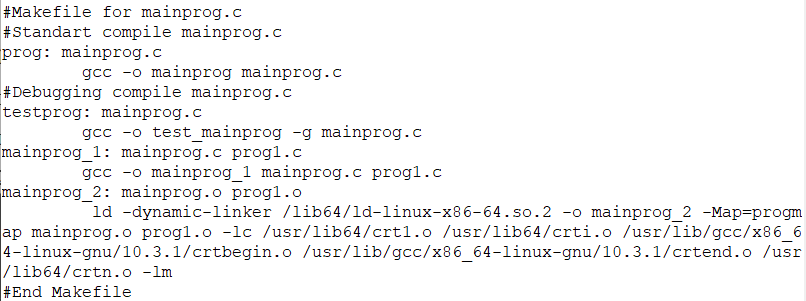


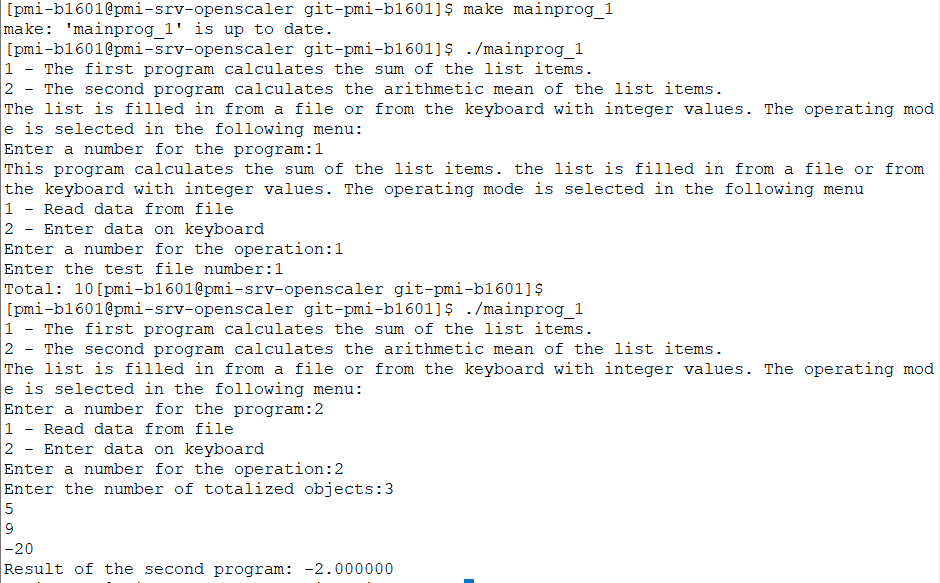


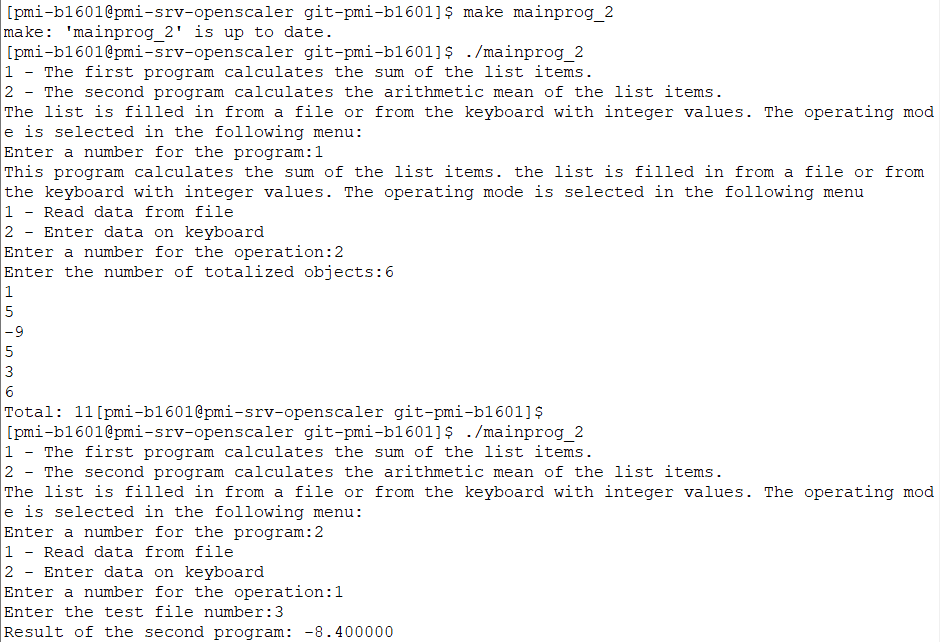


|  |  |  |  |
| --- | --- | --- | --- |
| Имя модуля (функции) | Исходный, Кбайт | Объектный, Кбайт | Машинный код, байт |
| prog1 | 2.4 | 4.3 | 823 |
| prog2 | 2.5 | - | - |
| mainprog | 4.3 | 5.6 | 1264 |

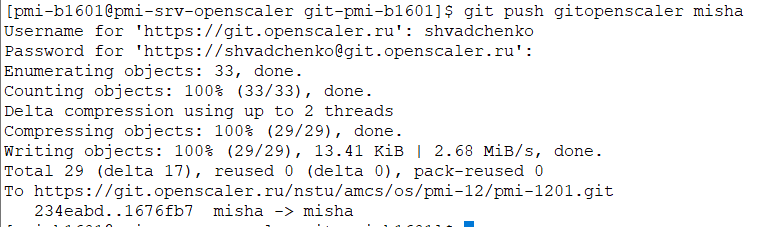
1. Добавьте в make-файл, разработанный при выполнении п. 1, два новых правила, реализующие п. 8 и 9 задания. Проверьте корректность его работы и сохраните его в репозитории.







1. Выполните экспорт всех данных из локального репозитория в центральный репозиторий, адрес которого будет указан преподавателем.



1. В ходе выполнения лабораторной работы были получены практические навыки по использованию инструментальных средств фонда свободного программного обеспечения при компиляции исходного кода, сборке, отладке и тестировании программ, написанных на языке С. Проработали контрольные вопросы по теме.