Задания для практических и лабораторных работ по курсу «Компьютерные сети и телекоммуникации»

Оглавление

1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4
1.1. Командные оболочки UNIX-подобных ОС 5
1.2. Файловая система ОС Linux 6
1.2.1. Типы файлов 7
1.2.2. Системные каталоги Linux 8
1.3. Управление терминалом 9
1.4. Основные команды управления файлами и каталогами 12
1.4.1. Текущий каталог, смена каталога 13
1.4.2. Список файлов, атрибуты файлов 14
1.4.3. Создание и удаление каталогов 16
1.4.4. Групповые символы 16
1.4.5. Создание, запись, просмотр содержимого файлов 18
1.4.6. Удаление, пересылка и копирование файлов 20
1.4.7. Изменение прав доступа к файлу 22
1.5. Принципы программирования в Linux 25
1.5.1. Использование зависимостей 38
1.5.2. Использование переменных и комментариев 39
2. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ 40
2.1. Лабораторная работа №1. Файловая система и основные команды управления файлами 41
2.2. Лабораторная работа №2. Принципы программирования в Linux 43
3. Конфигурирование абонентских VoIP шлюзов TAU-1M.IP 48
1. Цели и задачи лабораторной работы 48
2. Краткие теоретические сведения 49
2.1. Основные определения и понятия 49
2.2. Назначение 50
2.3. Характеристики устройства 51
2.4. Конструктивное исполнение 52
2.5. Сброс к заводским настройкам 56
2.6. План нумерации 56
3. Порядок выполнения лабораторной работы 59
3.1. Описание работы 59

3.2. Порядок выполнения работы 59
4. Конфигурирование Ethernet коммутаторов MES 64
4. Краткие теоретические сведения 65
4.1. Назначение коммутатора 65
4.2. Коммутаторы доступа серии MES2000 65
4.3. Основные технические характеристики 66
4.4. Конструктивное исполнение 67
4.5. Управление устройством. Режимы конфигурирования 69
5. Порядок выполнения лабораторной работы 71
5.1. Практическое задание №1. Базовое конфигурирование MES 71
5.2. Практическое задание №2. Конфигурирование Ethernet портов 75
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 79
ПРИЛОЖЕНИЯ 80
Приложение 1. Консольные текстовые редакторы 80

1. ОБЩИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Linux является *многопользовательской* и *многозадачной* операционной системой (ОС). Многозадачность означает, что система обеспечивает возможность параллельной обработки нескольких процессов. Многопользовательский режим означает, что в системе могут одновременно работать несколько пользователей, каждый из которых взаимодействует с ней через свой ***терминал***. Поэтому любой сеанс работы в ОС Linux начинается с ввода имени и пароля ранее зарегистрированного пользователя, либо регистрации нового.

Существует два режима работы в ОС Linux, определяющие способы входа в систему и взаимодействия с ней – текстовый (консольный) и графический. Традиционно базовым для UNIX-подобных систем является консольный режим работы. При этом пользователь вводит с клавиатуры команды и просматривает текстовую информацию на дисплее (часто такой режим работы называют «интерфейсом командной строки»). Понятия «клавиатура» и «дисплей» в данном случае во многом условны и не обязательно означают реальные устройства компьютера, на котором работает пользователь. Например, «дисплеем» может быть окно графической среды пользователя или интерфейс программы удаленного доступа, использующей протоколы *telnet* или *ssh*. Но при этом смысл от этого не изменяется: базовый интерфейс пользователя предполагает ввод команд и вывод текстовой информации. Для того, чтобы подчеркнуть «виртуальность» устройств ввода и вывода текста, их вместе называют терминалом. Во многих случаях (в частности, при настройке разного рода телекоммуникационного оборудования) администратору системы только такой режим работы и будет доступен. Кроме того, командная строка – самое удобное средство автоматизации рутинных процедур. Данное учебно-методическое пособие призвано помочь студентам освоить базовые навыки работы с командной строкой в ОС Linux.

1.1. Командные оболочки UNIX-подобных ОС

Для обеспечения интерфейса командной строки в операционных системах используется *командный интерпретатор* (или *командная оболочка* (англ. *shell*)) – компьютерная программа, обеспечивающая взаимодействие пользователя с системой в текстовом режиме, позволяя вводить команды. *Командой* в Linux принято считать все, что может быть исполнено: исполняемые файлы, встроенные команды оболочки, пользовательские функции, псевдонимы команд, файлы сценариев (скрипты) – наборы команд, собранные в текстовые файлы и объединенные некоторой общей целью [1, 2].

Командная оболочка обрабатывает вводимые пользователем команды и их аргументы, отправляет команды на исполнение, принимает возвращаемые значения и выполняет определённые действия в зависимости от этих значений. Кроме того, в командную оболочку встроен язык программирования (командный язык), необходимый для написания сложных разветвленных сценариев. Синтаксисом и функциональными возможностями командного языка и отличаются разные командные интерпретаторы UNIX-подобных ОС [2].

Для ОС Linux существует много командных оболочек. Некоторые из них представлены в Таблице 1.1.

Таблица 1.1. Разновидности командных оболочек ОС Linux

Имя исполняемого файла	**Описание**
sh	англ. *Bourne* ***sh****ell* – ранняя командная оболочка, разработанная С. Борном. Является фактически стандартом для многих UNIX-подобных ОС и присутствует практически в любом дистрибутиве Linux.

bash	англ. ***B**ourne **a**gain **sh**ell* – усовершенствованная модификация командной оболочки Bourne shell. Особенно популярна в среде Linux.
csh	англ. ***C sh**ell* – командный язык оболочки близок по синтаксису к языку Си.
tcsh	англ. ***T**ENEX **C sh**ell* – реализация csh с интерактивными возможностями, не уступающими bash. Удобна для интерактивной работы.
zsh	англ. ***Z sh**ell* – одна из современных командных оболочек. Добавлено большое количество улучшений Bourne shell.

В UNIX-подобных ОС пользователь может изменить командный интерпретатор, назначенный ему по умолчанию. В ОС Linux по умолчанию новому пользователю назначается командная оболочка **bash**.

1.2. Файловая система ОС Linux

Под ***файловой системой*** принято понимать логическую структуру, определяющую способ организации, хранения и именования данных на любых носителях информации. В отличие от семейства ОС Windows, во всех UNIX-подобных ОС файловая система представляет собой иерархическую (древовидную) структуру, растущую из одного *корневого каталога*. Корневой каталог обозначается символом «/», и этот же символ используется для разделения имен каталогов. Каждому пользователю Linux для хранения персональных файлов назначается также *«домашний» каталог*. Путь к домашнему каталогу пользователя `ivan` может выглядеть, например, так – `/home/ivan` (также домашний каталог может быть обозначен символом «~»).

Имена файлов и каталогов могут иметь длину до 255 символов. Linux различает регистр символов в именах файлов и каталогов. *Абсолютный путем к файлу* (или *полным именем файла*) называется список вложенных в

друг друга каталогов, начинающийся корневым каталогом, а заканчивающийся собственно именем файла. Любой другой путь к файлу называется *относительным*. Например, `/home/ivan/programs/main.c` – абсолютный путь к файлу `main.c`, находящемуся в подкаталоге `programs` домашнего каталога пользователя `ivan` (еще один способ – `~ivan/programs/main.c`).

В каждом каталоге есть два подкаталога, имеющих имена «точка» и «две точки». Первый служит указанием на данный каталог, а второй – на его «родительский» каталог. С помощью этих имен образуются относительные имена файлов. Так, именем вышеупомянутого файла `main.c` относительно домашнего каталога `/home/ivan` пользователя `ivan` будет `../programs/main.c`.

1.2.1. Типы файлов

С точки зрения UNIX-подобных ОС, каждый файл рассматривается как последовательность байтов. Такой подход упрощает организацию данных и обмен ими. В ОС Linux существуют следующие типы файлов:

- **обычный файл** (англ. *regular file*). Последовательность байтов (текстовые документы, программы, библиотеки и др.);
- **каталог** (англ. *directory*). Именованный набор ссылок на другие файлы и каталоги;
- **специальный файл физического устройства** (англ. *special device file*). Содержит данные, необходимые ОС для взаимодействия с физическими устройствами (жесткими дисками, принтерами и др.). По сути, специальный файл устройства является указателем на драйвер этого устройства. Различают файлы *символьных* (байт-ориентированных) устройств, для которых запись и чтение данных возможно сплошным потоком байтов, и *блочных* устройств, для которых запись и чтение данных осуществляется блоками фиксированной длинны;

- **символьная («мягкая») ссылка** (англ. *symbolic link*). Ссылка содержит путь к файлу, открываемому при обращении к ней. Целью ссылки может быть любой объект: файл, каталог, другая ссылка. Также в UNIX поддерживаются *жесткие* ссылки, являющиеся фактически дополнительным и равноправным именем для обычного файла;
- **именованный канал** (англ. *named pipe*). Данный файл также называется буфером FIFO (англ. *first in, first out*). Именованный канал позволяет различным процессам обмениваться данными;
- **сокет** (англ. *socket*). Файл, предназначенный для сетевого или межпроцессного обмена данными.

1.2.2. Системные каталоги Linux

В *системных каталогах* располагаются файлы, необходимые для настройки и администрирования системы, а также стандартные программы. Имена системных каталогов, их расположение и содержание практически одинаковы почти во всех дистрибутивах ОС Linux. В Таблице 1.2. представлены наиболее важные.

Таблица 1.2. Назначение основных системных каталогов

Каталог	Назначение
/(корневой)	Корневой каталог. Содержит все остальные файлы и каталоги.
/bin	Основные программы и утилиты, необходимые для работы в системе.
/sbin	Исполняемые файлы программ и утилит для системного администрирования.
/boot	Файлы, необходимые для загрузки системы (образ ядра).
/dev	Специальные файлы устройств.
/etc	Конфигурационные файлы системы и различных прикладных программ.
/lib	Системные библиотеки, необходимые для компиляции и

	запуска большинства программ.
/home	Домашние каталоги всех пользователей системы, кроме `root`.
/root	Домашний каталог суперпользователя.
/mnt **/media**	Точки монтирования для временно монтируемых файловых систем.
/var	Различные часто изменяющиеся файлы. Например, кэши различных программ, файлы системных журналов и др.
/tmp	Временные файлы.
/usr	Практически все остальное: исходные коды, программы, документация. Каталог для установки новых программ по умолчанию.

1.3. Управление терминалом

Как отмечалось ранее в Linux существует два вида интерфейсов взаимодействия с системой: *графический интерфейс пользователя* (англ. *Graphical user interface, GUI*) и *интерфейс командной строки* (англ. *Command Line Interface, CLI*).

Преимуществами графического интерфейса являются возможности визуального отображения программ и их содержимого, управления программами с помощью графических кнопок, всплывающих меню, окон и других элементов [4].

При использовании интерфейса командной строки управление программами осуществляется с помощью команд, вводимых с клавиатуры. Преимущества такого интерфейса: низкие требования к аппаратным ресурсам, легкость автоматизации выполнения команд, гибкость при составлении перечня действий из команд. Кроме того, в ряде случаев случае администратору системы только такой интерфейс и будет доступен [4].

Доступ к интерфейсу командной строки можно получить через *консоль* или *терминал*. При загрузке ОС[1] запускаются семь полноэкранных виртуальных консолей со своим независимым сеансом. Первые шесть консолей имеют интерфейс командной строки, в седьмой запускается графический интерфейс. Пользователь во время загрузки видит только графический режим [4].

Для переключения в одну из виртуальных консолей можно нажать сочетание клавиш **«Ctrl»+«Alt»+F{1-6}**. Для возврата в графический режим следует нажать сочетание клавиш **«Ctrl»+«Alt»+F7**.

Терминалом называется графическая программа эмулирующая консоль. Терминал позволяет выполнять команды, не выходя из графического интерфейса пользователя. Для запуска терминала следует нажать сочетание клавиш **«Ctrl»+«Alt»+T** или нажать клавишу **«Win»** и в открывшемся меню ввести слово «терминал». Внешний вид терминала показан на рис. 1.1.

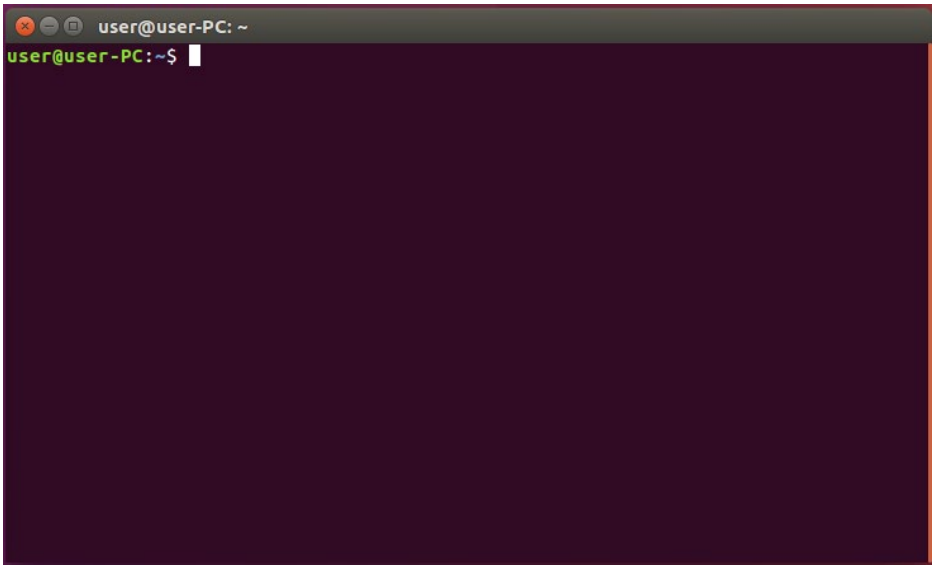


Рис. 1.1. Внешний вид терминала в ОС Ubuntu 16.04 LTS.

После запуска терминала пользователь видит строку с приглашением к вводу команд (см. рис. 1.1):

```
user@user-PC:~$
```

[1] Здесь и далее речь идет об ОС *Ubuntu 16.04 LTS* и программной оболочке *bash*.

где **user** – имя учетной записи пользователя, **@** – разделитель, **user-PC** – имя хоста, **:** – разделитель, **~** – указание на то, что текущим каталогом является домашний каталог пользователя **user** (см. пункт 1.4.1), **$** – символ приглашения к выполнению команды с правами *простого пользователя*.

По соображениям безопасности в UNIX-подобных ОС простые пользователи имеют ограниченный административный доступ к системе (в частности, не могут прерывать и изменять ход процессов, запущенных другими пользователями системы, копировать или изменять файлы в большинстве системных каталогов). Вместе с тем в ОС Linux обязательно есть один привилегированный пользователь – ***суперпользователь*** или ***root***. Полномочия суперпользователя в системе никак не ограничены. Этот пользователь имеет права на выполнение любых действий, удаление любых файлов и изменение любых параметров. Кроме того, большинство системных процессов работают от имени root [5].

Для получения root-доступа к выполнению команд используется утилита **sudo**. После ввода пароля строка с приглашением к вводу команд может выглядеть следующим образом:

```
root@user-PC:~#
```

где **#** – символ приглашения к вводу команд с правами суперпользователя.

Внимание!!! Будьте внимательны при выполнении команд с **sudo** или работе в сессии root-а. Вы безо всяких предупреждений со стороны системы сможете выполнить любую операцию, в частности, удалить системные файлы, сделав при этом систему неработоспособной.

Для упрощения работы с терминалом полезно запомнить ряд сочетаний клавиш (см. Таблицу 1.3) [4].

Таблица 1.3. Важные сочетания клавиш для работы в терминале

Клавиши	**Назначение**

«Ctrl»+«Shift»+C **«Ctrl»+«Shift»+V**	Копировать и вставить текст в терминале (также можно выполнить путем нажатия ПКМ и выбора соответствующего пункта).
«↑» («Ctrl»+P) **«↓» («Ctrl»+N)**	Прокрутка недавно использованных команд вверх и вниз.
«Enter»	Выполнение выбранной команды.
«Tab»	Крайне удобная возможность – ***автоподстановка*** команд и имён файлов. Если с выбранных символов начинается только одна команда, подставится именно она, а если их несколько, то по двойному нажатию **«Tab»** выведется список всех возможных вариантов.
«Ctrl»+R	Поиск по командам, введенным ранее. Если необходимо повторно выполнить очень длинную и сложную команду, можно ввести только её часть, а эта комбинация клавиш поможет найти команду целиком.

1.4. Основные команды управления файлами и каталогами

Синтаксис обычной команды в UNIX-подобных ОС имеет следующий вид:

`$ имя_команды [короткие_ключи] [длинные_ключи] [аргументы]`,

где в квадратных скобках указаны необязательные *параметры*. *Ключами* (или *модификаторами выполнения*) называются специальные символы, конкретизирующие или корректирующие действие команды. Ключи при записи команды можно располагать в любом порядке, разделяя их пробелами. Короткие ключи состоят из одной буквы и начинаются символом «-» (дефис). Длинные ключи – осмысленные слова, перед которыми стоит два знака «-». В качестве аргументов могут выступать имена файлов, числа, текст, спецсимволы, переменные. Аргументы также следует отделять друг от друга пробелами [2, 4].

Для выполнения команды можно ввести ее в терминале и нажать клавишу «Enter».

Внимание!!! Терминал чувствителен к регистру символов! Имена `User`, `user` и `USER` в Linux различаются!

Внимание!!! Большинство команд Linux обладают расширенным функционалом, задействовать который можно с помощью ключей. Кроме того, поведение команды может изменяться в зависимости от количества аргументов. Запомнить весь набор опций для множества команд невозможно, поэтому ОС снабжена обширной справочной информацией. Помощь предоставляется несколькими способами. Есть методы получения краткой информации и подробной многостраничной справки. Для получения справочной информации по использованию конкретной команды Linux, в большинстве случаев, необходимо запустить ее с ключом **`-h`** или **`--help`**. Более подробную информацию можно получить из комплекта документации, известного как **man-страницы** (англ. *manual* – руководство), поскольку он доступен по команде **`man`**. Например, получить справку об использовании команды **`uptime`** можно так:

```
$ uptime --help
```

или так:

```
$ man uptime
```

О том, как правильно использовать саму справочную систему, можно узнать по команде:

```
$ man man
```

К основным командам управления файлами и каталогами относятся команды перемещения по файловой системе, создания и удаления каталогов, копирования, перемещения, переименования и удаления файлов, управления атрибутами файлов и др. Далее рассмотрены некоторые из них.

1.4.1. Текущий каталог, смена каталога

В каждый момент времени с каждой выполняемой программой, в том числе с командным интерпретатором, связан единственный каталог, называемый *текущим*. От текущего каталога отсчитываются относительные пути к файлам. Для определения текущего каталога можно выполнить команду **pwd** без параметров [1, 2]:

```
$ pwd
/home/user
```

Для изменения текущего каталога можно ввести команду:

```
$ cd [путь_к_каталогу]
```

Например, для перемещения в каталог `/etc`, где хранится большинство конфигурационных файлов ОС Linux, можно выполнить команду:

```
$ cd /etc/
```

1.4.2. Список файлов, атрибуты файлов

Если необходимо просмотреть содержимое того или иного каталога, можно воспользоваться командой **ls**:

```
$ ls [путь_к_каталогу]
```

Без указания пути к каталогу команда **ls** выведет содержимое текущего каталога. Если указать ключ **-R**, то можно получить рекурсивный список, т.е. список всех подкаталогов директории, указанной в аргументе команды, и всех файлов, входящих в эти подкаталоги. Отобразить номера индексных дескрипторов[2] файлов можно с помощью ключа **-i**.

Для получения расширенной информации о файлах следует использовать ключ **-l**, например,

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 126584 мар  3  2017 /bin/ls
```

[2] **Индексный дескриптор** (*англ.* **inode**) – в UNIX-подобных ОС системная структура данных, содержащая метаинформацию о файлах, каталогах и других объектах файловой системы.

Обратите внимание, в качестве аргумента команды **ls** указан путь к конкретному файлу. Таким образом, вывод команды **ls** с ключом **-l** позволяет отобразить следующие *атрибуты* файла (справа налево):

- имя файла (в примере, **/bin/ls**);
- дата модификации файла (**мар 3 2017**);
- размер файла в байтах (**126584**);
- группа пользователей, которой принадлежит файл (**root**);
- пользователь, которому принадлежит файл (**root**);
- число жестких ссылок на файл (**1**);
- фрагмент, определяющий *права доступа* к файлу (**-rwxr-xr-x**).

Существует три права доступа к файлу: чтение (**r**), запись или модификация (**w**), выполнение (**x**). Для каталога право на выполнение означает право на просмотр содержимого. Отсутствие какого-либо права обозначается символом «**–**». Фрагмент (**-rwxr-xr-x**), определяющий права доступа к файлу, следует понимать следующим образом [6, 7]:

- первый символ – это тип файла («**–**» – обычный файл, «**d**» – каталог, «**l**» – символическая ссылка, «**b**» – специальный файл блочного устройства, «**c**» – специальный файл символьного устройства, «**p**» – именованный канал, «**s**» – сокет);
- последующие три символа (**rwx**) определяют *права доступа владельца файла*. Первый символ – это чтение, второй – запись, третий – выполнение. Таким образом, пользователю **root**, являющемуся владельцем файла **/bin/ls**, разрешено чтение, модификация и выполнение этого файла.
- следующие три символа (**r-x**) определяют *права доступа для членов группы владельца*. Пользователям, входящим в группу **root**, разрешено только читать и выполнять файл, а изменять его запрещено (средний символ – «**–**»).

- последние три символа (**r-x**) задают *права доступа для прочих пользователей*. Все остальные пользователи имеют право читать и выполнять файл.

Для изменения прав доступа к файлу используется команда **chmod** (см. пункт 1.4.7).

1.4.3. Создание и удаление каталогов

Для создания каталогов используется команда **mkdir** (англ. ***mk**e **dir**ectory*):

```
$ mkdir [имя_каталога]
```

При выполнении этой команды в новом каталоге автоматически создаются два подкаталога с именами «.» и «..» (указатели на созданный каталог и его родительский каталог соответственно). Например, с помощью команды:

```
$ mkdir emacs.src emacs.file
```

можно в текущем каталоге создать два каталога с именами `emacs.src` и `emacs.file`.

Для удаления пустого каталога можно воспользоваться командой:

```
$ rmdir [имя_каталога]
```

Если удаляемый каталог не существует или не пуст, **rmdir** сообщит об этом. Для удаления непустого каталога нужно вначале удалить все файлы, находящиеся в нем (см. пункт 1.4.6).

1.4.4. Групповые символы

Перед обсуждением команд для работы с файлами, полезно указать на особенность командной оболочки, позволяющей значительно повысить гибкость этих команд. Поскольку имена файлов используются в командной оболочке повсеместно, она поддерживает специальные символы, помогающие быстро определить группы имен файлов. Эти специальные

символы получили название *групповых символов* (англ. *wildcard pattern*) или символов подстановки (англ. *glob pattern*). Групповые символы позволяют создавать *шаблоны (маски)* имени, используя которые можно одной командой обратиться более чем к одному файлу. В Таблице 1.4. перечислены групповые символы и их соответствия [7]. Групповые символы можно использовать с любыми командами, принимающими имена файлов в виде аргументов.

Таблица 1.4. Групповые символы

Групповой символ	**Соответствие**
*****	Любая последовательность любых символов (включая пустую).
?	Любой одиночный символ.
[*символы*]	Любой один символ из указанного множества ***символов*** (пара символов, разделенных знаком «-», задает интервал символов).
[!*символы*]	Любой один символ, не принадлежащий указанному множеству ***символов***.
[[:*класс*:]]	Любой один символ, принадлежащий указанному ***классу***.

В Таблице 1.5 приведены некоторые часто используемые классы символов.

Таблица 1.5. Часто используемые классы символов

Класс символов	**Соответствие**
[:alnum:]	Любой алфавитно-цифровой символ.
[:alpha:]	Любой алфавитный символ.
[:digit:]	Любой цифровой символ.
[:lower:]	Любая буква в нижнем регистре.

[:upper:]	Любая буква в верхнем регистре.

В Таблице 1.6 приведены некоторые примеры шаблонов и их соответствия.

Таблица 1.6. Примеры использования групповых символов

Шаблон	**Соответствие**	**Соответствует**	**Не соответствует**
***.c**	Все имена файлов, заканчивающиеся на «.c».	test1.c a.c	test1.C a.c.b.doc
b*.txt	Все имена файлов, начинающиеся с символа «b», за которым следует любое число других символов, и заканчивающиеся на «.txt».	bworld.txt b.txt	rbworld.txt b.txt2
Data???	Все имена файлов, начинающиеся с символов «Data», за которыми следуют ровно три любых символа.	Data.cp Data1.c	Data.cpp Data2.cp
[abc]*	Все имена файлов, начинающиеся с символа «a», «b» или «c».	b.ext alpha.txt	Alpha text.c
test[0-9]	Все имена файлов, начинающиеся с символов «test», за которыми следуют одна цифра.	test0	Test1 test10
[[:upper]]*	Все имена файлов, начинающиеся с буквы в верхнем регистре	Source.cpp	test1.C
[![:digit:]]*	Все имена файлов, не начинающиеся с цифры	alpha.txt b11.ext	11.txt 1.c

1.4.5. Создание, запись, просмотр содержимого файлов

Для создания пустого файла можно воспользоваться командой **`touch`**:

```
$ touch [имя_файла или путь_к_файлу]
```

В качестве аргумента следует указать имя файла, если требуется создать файл в текущем каталоге, в противном случае – путь к файлу. Если в качестве аргумента команде **touch** передать имя существующего файла, она обновит время последнего изменения и доступа, не изменяя при этом содержимого файла [2].

Текстовые файлы можно создавать, вводя текст в командной строке, с помощью команды **cat**:

```
$ cat > file.txt
My file!
```

где «**>**» – символ перенаправления ввода-вывода. При этом команда **cat** считает своими входными данными поток байтов, поступающих с клавиатуры, и выводит их в файл `file.txt` [2]. Таким образом, все что будет введено с клавиатуры после этой команды будет записано в файл. Ввод заканчивается нажатием **«Ctrl»+D**.

Если файл с указанным именем уже существует, то команда **cat** перезапишет его. В том случае, когда необходимо добавить данные в конец файла, следует перенаправить вывод команды с помощью символа «**>>**»:

```
$ cat >> file.txt
```

Записать или дописать данные в файл можно также с помощью команды **echo**:

```
$ echo "My file!" > file.txt
```

Без символов перенаправления вывода **echo** выведет переданную ей строку в терминал:

```
$ echo My file!
My file!
```

Создавать и редактировать текстовые файлы в командной строке можно с помощью консольных текстовых редакторов, например, **vi/vim** или **nano** [7]. Функциональные возможности этих редакторов достаточно

обширны. Редактор **vi** (а также его улучшенная версия **vim**) известен своим сложным пользовательским интерфейсом, но при этом остается фактическим стандартом и гарантированно присутствует в любой UNIX-системе. Текстовый редактор **nano** менее универсален, но обладает значительно более интуитивным и удобным пользовательским интерфейсом. **nano** управляется сочетаниями клавиш. Запуск редактора осуществляется командой:

```
$ vi [имя_файла или путь_к_файлу]
```

или

```
$ nano [имя_файла или путь_к_файлу]
```

Если указанный файл существует, он будет открыт для редактирования, иначе редактор создаст новый файл с заданным именем.

Дополнительную информацию об использовании редакторов можно получить в Приложении 1 настоящего пособия, а также в источниках [6, 7], в документации на официальных сайтах [8, 9] и справочном руководстве **man**.

Для просмотра содержимого файла в окне терминала можно воспользоваться командой:

```
$ cat file.txt
```

Во многих случаях требуется выводить не все содержимое файла, а только некоторую его часть, например, некоторое количество строк с начала файла или с конца. В этом случае используются команды **head** и **tail**. По умолчанию, команда **head** выводит первые десять строк файла, а **tail** – последние десять [6]. Количество выводимых строк может быть изменено с помощью ключа **–n**. Например, вывести первые 15 строк системного журнала Linux `/var/log/syslog` можно командой:

```
$ head –n 15 /var/log/syslog
```

1.4.6. Удаление, пересылка и копирование файлов

Для удаления файлов и каталогов (наряду с **rmdir**) используется команда **rm**:

```
$ rm [элемент]
```

где ***элемент*** может быть именем файла, каталога или шаблоном.

Внимание!!! В UNIX-подобных ОС нет команды, отменяющей удаление. Будьте осторожны при удалении чего-либо с помощью команды **rm**. Данные при этом будут удалены окончательно. Поэтому всегда используйте команду **rm** с ключом **–i**:

```
$ rm -i file.txt
rm: удалить обычный файл 'file.txt'? y
```

В этом случае командная оболочка будет запрашивать у пользователя подтверждение операции перед удалением существующего файла. Далее следует ввести «y» или «n» для удаления файла или отмены операции соответственно.

Будьте особенно осторожны при использовании в аргументе команды **rm** групповых символов!!! Всякий раз проверьте вначале этот аргумент с командой **ls**. Это позволит увидеть, какие файлы будут удалены.

Для удаления каталогов необходимо использовать **rm** с ключом **–r**, например, для удаления каталога `/subfolder`, находящегося в текущем каталоге, со всем его содержимым можно использовать команду:

```
$ rm -ir subfolder
```

При этом будут рекурсивно удалены все подкаталоги `/subfolder`.

В Таблице 1.7 приведены некоторые примеры использования команды **rm** [7].

Таблица 1.7. Примеры использования команды **rm**

Команда	Результат
`$ rm file1.txt`	Удаление файла *file1.txt*.

$ rm -i file1.txt	Перед удалением *file1.txt* будет запрошено подтверждение операции.
$ rm -r file1.txt dir1	Удаление файла *file1.txt* и каталога *dir1* со всем его содержимым.
$ rm -rf file1.txt dir1	То же, что и выше, но в отсутствие *file1.txt* и/или *dir1* просто продолжит работу, не выводя никаких сообщений.

Команду **mv** можно использовать как для переименования, так и для перемещения файлов/каталогов. Например, переименуем файл `source.c` в `new_source.c`:

```
$ mv source.c new_source.c
```

Переместим файл `new_source.c` из текущего каталога в каталог `/home/temp`:

```
$ mv new_source.c ~/temp
```

Для копирования файлов применяется команда **cp** (в отличие от **mv** оригинальный файл при этом сохраняется). Например, команда

```
$ cp ~/temp/new_source.c copy_source.c
```

скопирует файл `new_source.c` из каталога `/home/temp` в текущий каталог и сохранит его под именем `copy_source.c`.

1.4.7. Изменение прав доступа к файлу

Определить права доступа к файлу/каталогу можно с помощью команды **ls** (см. пункт 1.4.2). Для изменения прав доступа к файлу/каталогу используется команда **chmod**. Существует два способа указания прав доступа: символьный (используя символы, задающие права доступа – r, w, x) и числовой (используя восьмеричные константы).

На наш взгляд более простым для понимания является числовой способ, при котором права доступа к файлу/каталогу для владельца, группы

владельца и прочих пользователей определяется с помощью трех восьмеричных цифр (см. Таблицу 1.8).

Таблица 1.8. Режимы доступа в восьмеричном представлении

Восьмеричное представление	Режим доступа	Восьмеричное представление	Режим доступа
0	---	4	r--
1	--x	5	r-x
2	-w-	6	rw-
3	-wx	7	rwx

Рассмотрим пример:

```
$ ls -l source.c
-rw-rw-r-- 1 user user 6 сен 26 14:44 source.c
$ chmod 700 source.c
$ ls -l source.c
-rwx------ 1 user user 6 сен 26 14:44 source.c
```

Передав команде **chmod** аргумент **700**, мы установили права для владельца **user**, позволяющие ему читать данные из файла, записывать их файл, запускать файл на исполнение, а всем прочим пользователям отобрали все права.

Несмотря на кажущееся неудобство необходимости запоминания соответствий между восьмеричными константами и режимами доступа, вам, скорее всего, придется использовать лишь несколько популярных шаблонов:

- 644 – владельцу можно читать и изменять файл, остальным пользователям – только читать;
- 666 – читать и изменять файл можно всем пользователям;
- 777 – всем можно читать, изменять и выполнять файл.

С символьным способом задания прав доступа читателю предлагается ознакомиться самостоятельно, например, по [6, 7] или по man-странице

команды **chmod**. Укажем лишь на часто используемый символьный аргумент, позволяющий сделать файл исполнимым:

```
$ chmod +x script
```

1.5. Принципы программирования в Linux

Компилятор

Исходный программный код большинства утилит Linux, а также многих приложений для этой системы написан на языке C или C++.

Программы на языке C создаются традиционным для разработки способом: вы пишете программу, компилируете ее, а затем запускаете. То есть, когда вы пишете программу на языке C и желаете ее запустить, вы должны *скомпилировать* исходный код, превратив его в низкоуровневое двоичное представление, которое понимает компьютер.

Исполняемый файл компилятора C в большинстве версий систем Unix является компилятором GNU C, gcc, хотя новый компилятор clang, разработанный группой LLVM, набирает популярность. Файлы программного кода на языке C имеют расширение .c. Рассмотрим одиночный модульный файл hello.c с кодом на языке C [], который можно найти в книге Брайана У. Кернигана (Brian W. Kernighan) и Денниса М. Ритчи (Dennis M. Ritchie) *The C Programming Language* («Язык программирования C»), 2-е издание (Prentice Hall, 1988):

```
#include <stdio.h>
main() {
printf("Hello, World.\n");
}
```

Поместите этот код в файл с названием `hello.c`, а затем запустите такую команду:

```
$ gcc hello.c
```

В результате появится исполняемый файл с именем `a.out`, который можно запустить подобно любому другому исполняемому файлу системы. Однако следует присвоить этому исполняемому файлу другое имя (например, hello). Чтобы это сделать, используйте параметр компилятора `-o`:

```
$ gcc -o hello hello.c
```

У GCC великое множество опций, которые описаны в ***man gcc***.

Для небольших программ компилировать больше нечего. Может понадобиться добавить каталог включаемых файлов или библиотеку.

Большинство программ на языке C слишком большие, чтобы уместиться в пределах единственного файла с исходным кодом. Исполинские файлы становятся неуправляемыми для программиста, а компиляторы иногда даже испытывают сложности при синтаксическом анализе больших файлов. По этой причине разработчики группируют компоненты исходного кода вместе, предоставляя каждому фрагменту отдельный файл.

При компиляции большинства файлов .c исполняемый файл создается не сразу. Вместо этого сначала используется параметр компилятора `-c` для каждого файла, чтобы создать объектные файлы. Чтобы понять, как это устроено, предположим, что у вас есть два файла, `main_file.c` и `file_2.c`. Следующие две команды для компилятора выполняют основную часть работы по созданию программы:

```
$ cc -c main_file.c
$ cc -c file_2.c
```

Объектный файл является двоичным файлом, который процессор уже почти готов понять, если учесть еще несколько моментов. Во-первых, операционная система не знает, как запускать объектные файлы, а во-вторых, вам, вероятно, потребуется скомбинировать несколько объектных файлов и системных библиотек, чтобы создать завершенную программу.

Чтобы создать полностью функционирующий исполняемый файл из одного или нескольких объектных файлов, следует запустить компоновщик, команду `ld` в `Unix`. Программисты редко используют эту команду в командной строке, поскольку компилятор C знает, как запускать компоновщик. Для создания исполняемого файла с названием `myprog` из двух приведенных выше объектных файлов запустите такую команду:

```
$ cc -o myprog main_file.c file_2.c
```

Отследить правильный включаемый файл не всегда легко. Иногда несколько включаемых файлов с одинаковыми именами расположены в разных каталогах и неясно, какой из них правильный. Когда компилятор не может обнаружить включаемый файл, сообщение об ошибке выглядит так:

```
badinclude.c:1:22: fatal error: file_1.h: No such file or directory
```

Это сообщение говорит о том, что компилятор не может найти заголовочный файл `file_1.h`, на который ссылается файл `badinclude.c`. Эта ошибка является прямым следствием такой директивы в первой строке файла `badinclude.c`:

```
#include < file_1.h >
```

По умолчанию в `Unix` каталогом для включаемых файлов является `/usr/include`; компилятор всегда просматривает его, если вы явно не укажете ему не выполнять этого. Тем не менее, можно настроить компилятор так, чтобы он просматривал другие каталоги (большинство каталогов с заголовочными файлами содержит слово `include` где-либо в своем имени).

Следует также опасаться включаемых файлов, использующих двойные кавычки (" ") вместо угловых скобок (< >), например так:

```
#include "myheader.h"
```

Двойные кавычки означают, что заголовочный файл не располагается в системном каталоге для включаемых файлов и компилятору следует поискать его путь. Часто это говорит о том, что включаемый файл находится

в том же каталоге, что и файл с исходным кодом. Если вам встретится проблема с двойными кавычками, то, вероятно, вы пытаетесь скомпилировать неполный исходный код.

Компилятор C знает о вашей системе недостаточно для того, чтобы самостоятельно создать пригодную программу. Для построения завершенных программ вам необходимы ***библиотеки***. Библиотека C является набором распространенных, заранее скомпилированных функций, которые можно встраивать в программу. Например, многие исполняемые файлы используют библиотеку `math`, поскольку она обеспечивает работу с тригонометрическими и другими математическими функциями.

Библиотеки вступают в игру главным образом во время компоновки, когда программа-компоновщик создает исполняемый файл из объектных файлов. Например, если у вас есть программа, которая использует библиотеку `gobject`, но вы забыли указать компилятору о связывании с этой библиотекой, то появятся ошибки компоновщика, подобные этой:

```
badobject.o(.text+0x28): undefined reference to
'g_object_new'
```

Наиболее важные части этого сообщения выделены жирным шрифтом. Когда компоновщик проверял объектный файл `badobject.o`, ему не удалось найти функцию, которая выделена жирным шрифтом, и, как следствие, не удалось создать исполняемый файл. В данном частном случае можно предположить, что вы забыли о библиотеке `gobject`, поскольку отсутствующая функция называется `g_object_new()`.

Если на экране появились сообщения об ошибках, введите `gcc -Wall -o errorlog file1.c`, чтобы получить дополнительную информацию. Затем в текущем каталоге откройте файл «`errorlog`». Для этого введите `cat errorlog`.

Отладчик

Стандартным отладчиком в системах Linux является gdb; доступны также системы с дружественным к пользователю интерфейсом, например Eclipse IDE и Emacs. Чтобы включить полную отладку ваших программ, запустите компилятор с параметром **-g** для записи таблицы имен и другой отладочной информации в исполняемый файл. Чтобы запустить отладчик gdb для исполняемого файла program, выполните такую команду:

```
$ gdb program
```

Вы должны получить приглашение (gdb). Чтобы запустить программу program с параметром командной строки options, введите следующую команду после приглашения отладчика:

```
(gdb) run options
```

Если программа в порядке, она должна запускаться, работать и завершать выполнение нормально. Однако если возникает проблема, отладчик gdb останавливается, выводит ошибочный исходный код и возвращает вас в строку приглашения (gdb). Поскольку фрагмент исходного кода часто содержит подсказку о причине проблемы, вам может потребоваться вывести значение какой-либо переменной, с которой связана ошибка. Команда print работает также для массивов и структур языка C.

```
(gdb) print variable
```

Чтобы отладчик остановил программу в указанном месте исходного кода, используйте контрольные точки. В следующей команде файл *file* является файлом с исходным кодом, а параметр *line_num* — это номер строки этого файла, в которой отладчик должен остановиться:

```
(gdb) break file:line_num
```

Для продолжения отладки выполните такую команду:

```
(gdb) continue
```

Чтобы удалить контрольную точку, введите команду:

(gdb) **clear** *file:line_num*

В данном пособии описана малая часть возможностей отладчика для получения подробной информации воспользуйтесь функцией man.

Принципы модульного программирования

Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

Модуль – функционально законченный фрагмент программы. Под модулем в программировании понимается сгруппированные по своему функциональному назначению совокупности констант, типов данных, функций обрабатывающих эти данные, независимые от основной программы, компилируемые отдельно и допускающие повторное использование.

Исходные тексты совокупности функций для решения какой-либо подзадачи, как правило, размещаются в отдельном модуле (файле). Такой файл называют исходным (sources). Обычно он имеет расширение .c или .cpp Прототипы всех функций исходного файла выносят в отдельный так называемый заголовочный файл (header file), для него принято использовать расширение .h или .hpp.

Таким образом, заголовочный файл x1.h содержит интерфейс для некоторого набора функций, а исходный файл x1.cpp содержит реализацию этого набора. Если некоторая функция из указанного набора вызывается из какого-то другого исходного модуля x2.cpp, то необходимо обязательно включить в этот модуль заголовочный файл x2.h с помощью директивы #include. Негласное правило стиля программирования на C++ требует

включения этого же заголовочного файла (с помощью `#include`) и в исходный файл `x1.cpp`.

Хотя использование глобальных переменных считается дурным тоном в программировании, иногда возникают ситуации, когда невозможно избежать их использования. В многофайловом проекте возможны два «вида глобальности переменных». Если некоторая глобальная переменная `glvar1` объявлена в файле `x1.cpp` с модификатором `static`, то она видима от точки определения до конца этого файла, то есть области видимости ограничена файлом. Если же другая глобальная переменная `glvar2` объявлена в файле `x1.cpp` без модификатора `static`, то она может быть видимой в пределах всего проекта. Правда, для того, чтобы она оказалась видимой в другом файле, необходимо в этом файле ее объявление с модификатором extern (рекомендуется это объявление поместить в файл `x.h`).

В заголовочном файле принято размещать:

- определения типов, задаваемых пользователем, констант, шаблонов;
- объявления (прототипы) функций;
- объявления внешних глобальных переменных (с модификатором extern);
- пространства имен.

При использовании заголовочных файлов иногда возникает проблема их повторного включения. Проблема может возникнуть при иерархическом проектировании структур данных, когда в некоторый заголовочный файл `x2.h` включается при помощи директивы `include` другой заголовочный файл `x1.h` (например, для использования типов, определенных в этом файле) и где-нибудь еще, например, в основной программе.

Для решения этой проблемы рекомендуется использовать так называемые стражи включения. Данный способ состоит в следующем: чтобы предотвратить заголовочных файлов, содержимое каждого `.h` файла должно

находиться между директивами условной компиляции #ifndef #endif, а внутри устанавливаться признак включения при помощи директивы #define.

```
#ifndef ИМЯ_ЗАГОЛОВОЧНОГО_ФАЙЛА
#define ИМЯ_ЗАГОЛОВОЧНОГО_ФАЙЛА

/* здесь помещается остальной текст заголовочного файла */
 /* ИМЯ_ЗАГОЛОВОЧНОГО_ФАЙЛА: */
#endif
```

Например, заголовочный файл myFunctions.h, в котором размещены объявления функций f и g, будет выглядеть так:

```
#ifndef MY_FUNCTIONS_H
#define MY_FUNCTIONS_H

void f(int n);
double g(double a, double b);
 /* MY_FUNCTIONS_H: */
#endif
```

Утилита make

Хотя и возможно скомпилировать несколько исходных файлов вручную, как показано в примере выше, трудно отслеживать их во время компиляции, если число таких файлов велико. Утилита **make** является стандартом Unix для управления компиляцией. Make - утилита, автоматизирующая процесс преобразования файлов из одной формы в другую

Главной идеей утилиты make является ***цель*** – то, чего вы желаете достичь. Целью может быть файл (файл .o, исполняемый файл и т. д.) или ярлык. Кроме того, некоторые цели зависят от других целей; например, вам

необходимо создать полный набор файлов .o, прежде чем вы сможете скомпоновать исполняемый файл. Эти требования называются ***зависимостями***.

Чтобы собрать цель, утилита `make` следует какому-либо правилу, например, определяющему, как перейти от исходного файла .c к объектному файлу `.o`. Утилите `make` уже известны некоторые правила, но вы можете изменить их, а также создать собственные. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки.

Утилита `make` автоматически определяет, какие части большой программы должны быть перекомпилированы, и выполняет необходимые для этого действия.

Утилита использует специальные `make`-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения.

На основе информации о времени последнего изменения каждого файла `make` определяет и запускает необходимые программы.

Перед тем, как использовать `make`, вы должны создать так называемый `make`-файл (`Makefile`).

Обычно, имеет смысл давать своему `make`-файлу имя `makefile`, либо `Makefile`.

Если вы хотите использовать нестандартное имя для вашего `make`-файла, вы можете указать его в командной строке, используя опции `-f` или `-file`.

`Make`-файлы состоят из 5 составляющих: *явных правил*, *неявных правил*, *определений переменных*, *директив и комментариев*.

- *Явное правило* сообщает, когда и как нужно преобразовать один или несколько файлов, которые называются целевыми правилами. В явном правиле перечисляются и другие файлы, от которых зависят цели,

называемые зависимостями цели, а также указывается способ, используемый для создания или обновления цели.

• *Негласное правило* сообщает, когда и как преобразовать класс файлы, исходя из их названия. Оно содержит описание, как цель может влиять на файлы с именем, похожим на цель, и указывает способ для создания или обновления такой цели.

• *Определение переменной* является строкой, в которой указывается текстовое строковое значение переменной, которая может быть подставлена в текст позже.

• *Директива* указывает утилите make на выполнение чего-либо особенного при чтении файла `makefile` (например, чтение другого `make`-файла).

• Комментарии в строке `makefile` начинаются с символа `'#'`. Сам комментарий и остальная часть строки игнорируется.

Простой `make`-файл состоит из "правил" (rules) следующего вида:

```
цель: зависимости
    системные команды
```

Целью обычно является имя файла, который генерируется программой; примеры целей — исполняемые или объектные файлы. Цель может также являться именем действия, которое необходимо выполнить, например, «`clean`».

Зависимость (также называется предварительным условием) – это файл, используемый в качестве входных данных для создания цели. Цель часто зависит от нескольких файлов. Однако правило, указывающее способ для цели, не должно иметь любых предварительных условий. Например, правило, содержащее команду `delete` («удалить»), связано с целью `clean` («очистить»), не имеющей предварительного условия.

Системная команд(ы) (также называемая способом) — это действие, которое выполняет утилита `make`. Способ может содержать более одной

команды, либо в той же строке или каждая в своей строке. Обычно, цель (target) представляет собой имя файла, который генерируется в процессе работы утилиты make.

В правиле может содержаться несколько команд - каждая на свое собственной строке.

Важное замечание: строки, содержащие команды обязательно должны начинаться с символа табуляции!

По умолчанию, make начинает свою работу с первой встреченной цели (кроме целей, чье имя начинается с символа <.>).

Эта цель будет являться главной целью по умолчанию (default goal).

Главная цель (goal) - это цель, которую стремится достичь make в качестве результата своей работы.

Таким образом, когда вы даете команду make, утилита make читает make-файл из текущей директории и начинает его обработку с первого встреченного правила сверху вниз.

```
all:
g++ main.cpp hello.cpp factorial.cpp -o hello
```

Список некоторых стандартных целей:

all — является стандартной целью по умолчанию. При вызове make ее можно явно не указывать.

clean — очистить каталог от всех файлов полученных в результате компиляции.

install — произвести инсталляцию приложения в систему.

uninstall — и деинсталляцию приложения соответственно.

Компилируемые и make файлы должны находиться в одном каталоге, так как это показано на рис.1.

Содержимое компилируемых файлов показано на рис. 2-5.

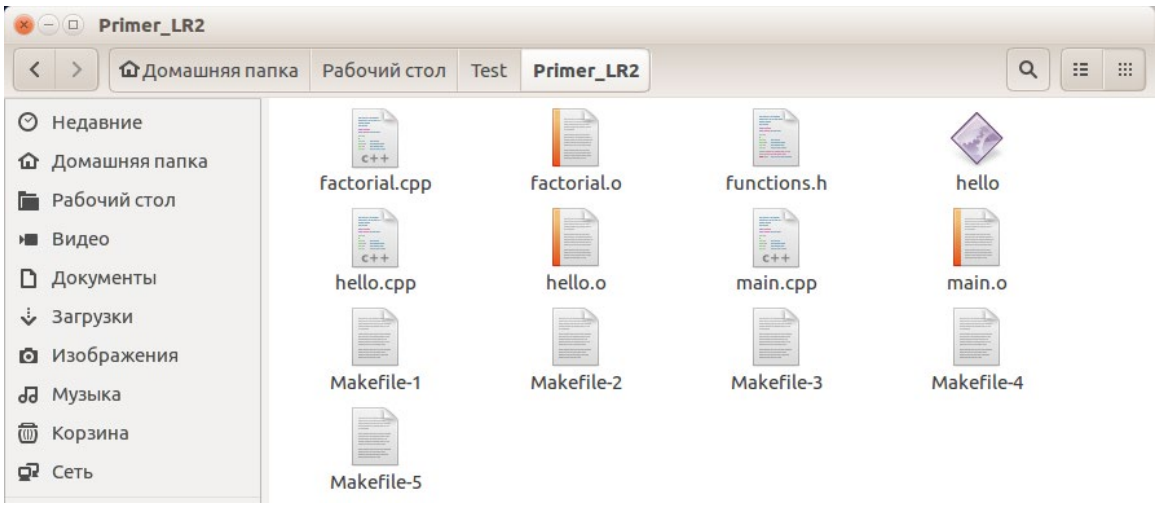


Рис.1. Содержимое каталога после компиляции

Рис.2. Файл main.cpp

Рис.3. Файл factorial.cpp

Рис.4. Файл hello.cpp

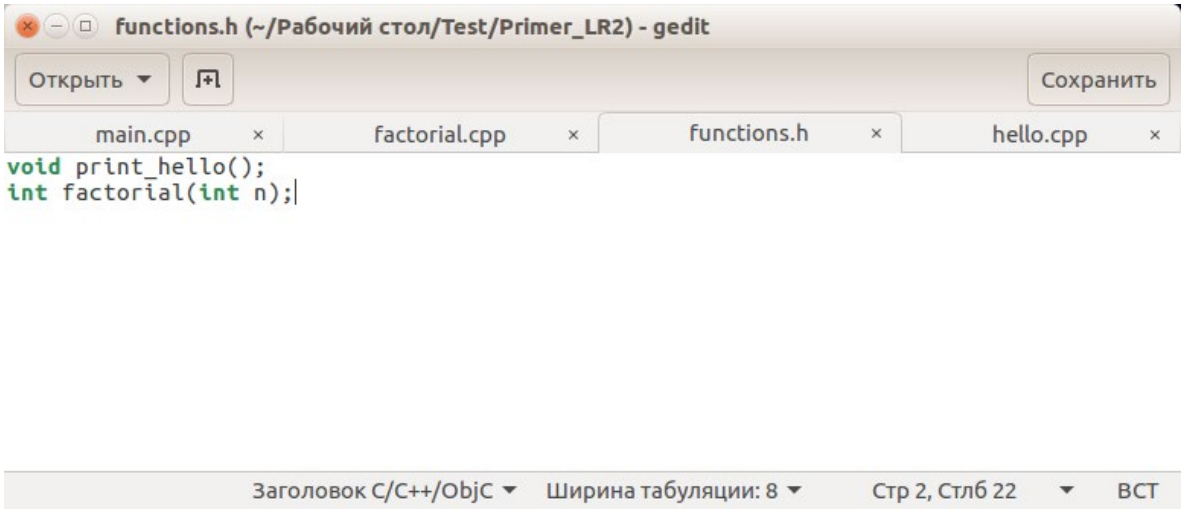


Рис.5. Файл functions.h

Если запустить `make` то программа попытается найти файл с именем по умолчание `Makefile` в текущем каталоге и выполнить инструкции из него. Если в текущем каталоге есть несколько `Makefile`, то можно указать на нужный вот таким образом:

```
make -f MyMakefile.
```

Компилятор берет файлы с исходным кодом и получает из них объектные файлы. Затем линковщик берет объектные файлы и получает из них исполняемый файл. **Сборка = компиляция + линковка.**

Пример Makefile

Для нашего файлов, содержимое которых показано на рис. 2-5 Makefile будет выглядеть так:

```
all:
	g++ main.cpp hello.cpp factorial.cpp -o hello
```

Обратите внимание, что строка с командой должна начинаться с табуляции!

Сохраните это под именем `Makefile-1` в каталоге с проектом и запустите сборку командой:

```
make -f Makefile-1
```

В примере цель называется `all`. Это цель по умолчанию для Makefile, которая будет выполняться, если никакая другая цель не указана явно. Также у этой цели в этом примере нет никаких зависимостей, так что make сразу приступает к выполнению нужной команды. А команда в свою очередь запускает компилятор.

1.5.1. Использование зависимостей

Использовать несколько целей в одном Makefile полезно для больших проектов. Это связано с тем, что при изменении одного файла не понадобится пересобирать весь проект, а можно будет обойтись пересборкой только измененной части. Пример:

```
all: hello
hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello
main.o: main.cpp
	g++ -c main.cpp
factorial.o: factorial.cpp
	g++ -c factorial.cpp
hello.o: hello.cpp
	g++ -c hello.cpp
```

```
clean:
        rm -rf *.o hello
```

Этот текст надо сохранить под именем `Makefile-2` все в том же каталоге.

Теперь у цели `all` есть только зависимость, но нет команды. В этом случае `make` при вызове последовательно выполнит все указанные в файле зависимости этой цели. Еще добавилась новая цель `clean`. Она традиционно используется для быстрой очистки всех результатов сборки проекта. Очистка запускается так: `make -f Makefile-2 clean`.

Поясним синтаксис последней строки **rm** – команда для удаления файлов.

Команда rm служит для удаления указанных имен файлов из каталога. Если заданное имя было последней ссылкой на файл, то файл уничтожается. Для удаления пользователь должен обладать правом записи в каталог; иметь право на чтение или запись файла не обязательно. Следует заметить, что при удалении файла в Linux, он удаляется навсегда. Здесь нет возможностей вроде "мусорной корзины" в Windows.

Так что, если файл удален, то он удален!

Если нет права на запись в файл и стандартный ввод назначен на терминал, то выдается (в восьмеричном виде) режим доступа к файлу и запрашивается подтверждение; если оно начинается с буквы **y**, то файл удаляется, иначе - нет. Если стандартный ввод назначен не на терминал, команда rm ведет себя так же, как при наличии опции -f.

1.5.2. Использование переменных и комментариев

Переменные широко используются в Makefile. Например, это удобный способ учесть возможность того, что проект будут собирать другим компилятором или с другими опциями.

```
# Это комментарий, который говорит, что переменная CC указывает компилятор, используемый для сборки
CC=g++
#Это еще один комментарий. Он поясняет, что в переменной CFLAGS лежат флаги, которые передаются компилятору
CFLAGS=-c -Wall

all: hello

hello: main.o factorial.o hello.o
	$(CC) main.o factorial.o hello.o -o hello
main.o: main.cpp
	$(CC) $(CFLAGS) main.cpp
factorial.o: factorial.cpp
	$(CC) $(CFLAGS) factorial.cpp
hello.o: hello.cpp
	$(CC) $(CFLAGS) hello.cpp
clean:
	rm -rf *.o hello
```

Это Makefile-3

Переменные – очень удобный инструмент. Для их использования надо просто присвоить им значение до момента их использования. После этого можно подставлять их значение в нужное место вот таким способом: $(VAR).

2. ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

2.1. Лабораторная работа №1. Файловая система и основные команды управления файлами

ЦЕЛЬ РАБОТЫ

Ознакомление с ОС *Linux*, получение практических навыком работы с командным интерпретатором *bash*, изучение принципов организации файловой системы и базовых команд управления файлами в ОС *Linux*.

ИЗУЧАЕМЫЕ КОМАНДЫ

cat, cd, cp, cut, echo, find, grep, head, ln, ls, mkdir, mv, paste, pwd, sort, tail, tar, touch, uniq, wc

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Внимание!!! При выполнении данной работы все вводимые команды и отображаемые командной оболочкой результаты должны быть занесены в отчет.

1. Запустить терминал. Просмотреть имя текущего каталога с помощью команды **`pwd`**. Отобразить рекурсивный список содержимого текущего каталога с помощью команды **`ls`**.

2. Перейти в корневой каталог с помощью команды **`cd`**. Отметить, как изменилось строка приглашения. Отобразить содержимое корневого каталога.

3. Перейти в домашний каталог. Создать директорию `test` с помощью команды **`mkdir`**. Создать поддиректорию `subtest` в директории `test`. Отобразить рекурсивный список содержимого домашнего каталога.

4. Создать в директории `test` пустой файл `first.txt` с помощью команды **`touch`**.

5. С помощью консольного текстового редактора ввести в файл `first.txt` текст из 5-ти строк следующего вида (строка начинается с цифры – порядкового номера строки):

   1) my name is …

2) my surname is …

3) login is …

4) <текст произвольного содержания (не менее 2-х слов)>

5) <текст произвольного содержания (не менее 2-х слов)>

6. Добавить строку текста **“6) My first file is first.txt”** в конец файла `first.txt` помощью команды **`echo`**.

7. Убедиться, что файл создан, определить его размер, номер индексного дескриптора, права доступа (команда **`ls`**). Отразить эти данные в отчете. Просмотреть содержимое файла с помощью команды **`cat`**.

8. С помощью команды **`cp`** скопировать файл `first.txt` в каталог `/home/test/subtest` и сохранить его под именем `copy.txt`. Переименовать файл `first.txt` в `orig.txt` с помощью команды **`mv`**. Определить номера индексных дескрипторов файлов `copy.txt` и `orig.txt`.

9. Изменить права доступа к файлам `copy.txt` и `orig.txt`: владельцу все права, группе только чтение, остальным никаких прав доступа. Отразить новые права доступа этих файлов с помощью команды **`ls`**.

10. Создать файл `orig_2.txt`. Занести в него

2.2. Лабораторная работа №2. Принципы программирования в Linux

1. Подключить `include`-файл `stdio.h`, в функции `main` обеспечить вывод строки, например, «Привет мир, я Ваше имя» и вызов функции `sleep` с параметром 10.

2. Сохранить файл и откомпилировать программу с помощью `gcc` или `g++`.

3. Изучить сообщения компилятора и исправить ошибки (для вызова функции `sleep` необходим `include`-файл `unistd.h`).

4. Перекомпилировать программу, при отсутствии ошибок запустить на выполнение.

5. Изменить значение параметра функции `sleep`, перекомпилировать программу, запустить и оценить изменения.

6. Создать программу, состоящую минимум из трех файлов для реализации задания:

a) Определить функцию, возвращающую расстояние между точками.
   Выполнить перегрузку функции для следующих типов параметров:
   Два параметра типа структура «точка» (координаты x, y).
   Четыре параметра типа `float`.
   Четыре параметра типа `double`.

b) Определить функцию, возвращающую следующую минуту.
   Выполнить перегрузку функции для следующих типов параметров:
   Структура «время» (часы, минуты, секунды).
   Три целочисленных параметра: часы, минуты, секунды.
   Два целочисленных параметра: часы, минуты.

c) Определить функцию, находящую произведение ненулевых элементов массива. Выполнить перегрузку функции для следующих типов параметров:
   Одномерный массив типа `int` размерностью `N`.

Одномерный массив типа `float` размерностью `N`.

Одномерный массив типа `double` размерностью `N`.

d) Определить функцию, возвращающую прошедшее время в минутах (считать, что разница между передаваемыми значениями не превышает 24 часа). Выполнить перегрузку функции для следующих типов параметров:

Два параметра типа структура «время» (часы, минуты, секунды).

Шесть целочисленных параметра: часы, минуты, секунды.

Четыре целочисленных параметра: часы, минуты.

e) Определить функцию, возвращающую минимальное из нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:

Два параметра типа `int`.

Три параметра типа `int`.

Два параметра типа `float`.

Три параметра типа `float`.

Три параметра типа `double`.

f) Определить функцию, возвращающую количество дней с начала года. Выполнить перегрузку функции для следующих типов параметров:

Структура «дата» (год, месяц, день).

Три целочисленных параметра: год, месяц, день.

Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

g) Определить функцию, возвращающую n-ю степень числа x. Выполнить перегрузку функции для следующих типов параметров:

Два параметра: x и n – оба типа `int`.

Два параметра: x и n – оба типа `float`.

Два параметра: x – типа `float`, и n – типа `int`.

h) Определить функцию, находящую минимальный элемент массива. Выполнить перегрузку функции для следующих типов параметров:

Одномерный массив типа `int` размерностью N.

Одномерный массив типа `float` размерностью N.

Одномерный массив типа `double` размерностью N.

i) Определить функцию, возвращающую количество минут с начала суток. Выполнить перегрузку функции для следующих типов параметров:

Структура «время» (часы, минуты, секунды).

Три целочисленных параметра: часы, минуты, секунды.

Два целочисленных параметра: часы, минуты.

j) Определить функцию, возвращающую среднеарифметическое нескольких чисел. Выполнить перегрузку функции для следующих типов параметров:

Два параметра типа `int`.

Три параметра типа `int`.

Три параметра типа `float`.

Два параметра типа `double`.

k) Определить функцию, возвращающую день недели (иметь в виду, что 1 января 1-го года нашей эры было понедельником). Выполнить перегрузку функции для следующих типов параметров:

Структура «дата» (год, месяц, день).

Три целочисленных параметра: год, месяц, день.

Два целочисленных параметра: месяц, день (считать передаваемые числа датой текущего года).

l) Определить функцию, находящую среднеарифметическое элементов массива. Выполнить перегрузку функции для следующих типов параметров:

Одномерный массив типа `int` размерностью N.

Одномерный массив типа `float` размерностью N.

Одномерный массив типа `double` размерностью N.

3. Конфигурирование абонентских VoIP шлюзов TAU-1M.IP

1. Цели и задачи лабораторной работы

Во время выполнения лабораторной работы по «Конфигурирование абонентских VoIP шлюзов TAU-1M.IP» преследуются следующие цели:

- изучение способов управления и функционала устройства;
- получение практических навыков настройки и эксплуатации оборудования;
- изучение возможностей оборудования при выполнении различных задач;
- создание различных схем и вариантов подключения изучаемого оборудования с возможностью тестирования интересующих конфигураций.

2. Краткие теоретические сведения

2.1. Основные определения и понятия

Процесс конфигурирования и эксплуатация абонентских шлюзов TAU подразумевает знание основных теоретических сведений. Далее приводятся основные определения и понятия необходимые для дальнейшей работы.

Абонентский шлюз — устройство, предназначенное для передачи голосовой и факсимильной информации через сети передачи данных (IP-сети).

Порты FXS (Foreign eXchange Station/Subscriber) — аналоговый интерфейс эмулирующий расширение интерфейса АТС (Автоматической Телефонной Станции). В качестве FXS интерфейса/порта может выступать интерфейс любого оборудования, которое со стороны телефонного аппарата является АТС. Например, для подключения телефона к VoIP шлюзу TAU необходим один свободный FXS порт.

Порты FXO (Foreign eXchange Office) — аналоговый интерфейс абонентских устройств для подключения к телефонной станции. Примером таких устройств могут служить телефонные аппараты, факсы, модемы.

PBX, УПАТС, мини-АТС (Private Automatic Branch eXchange) — автоматическая телефонная станция, предназначенная для эксплуатации в пределах организации.

SIP (Session Initiation Protocol) — протокол передачи данных, описывающий способ установления и завершения пользовательского сеанса, включающего обмен мультимедийным содержимым, например, для IP-телефонии, сервиса мгновенных сообщений, аудио- и видеоконференций. Протокол описывает, каким образом клиентское приложение может запросить начало соединения у другого клиента, используя его уникальное имя. Протокол определяет способ согласования между клиентами об открытии каналов обмена на основе других протоколов, которые могут использоваться для непосредственной передачи информации.

SIP-T (Session Initiation Protocol for Telephones) — модификация протокола SIP для взаимодействия с телефонными сетями общего пользования (ТфОП). Суть модификации заключается в «прозрачной» передаче сообщений ISUP по IP-сетям.

H.323 — рекомендация ITU-T, определяющая набор стандартов для передачи мультимедийных данных по сетям с пакетной передачей (IP-сетям). В рекомендации существуют такие типы устройств как терминал, шлюз,

привратник, MCU (сервер многосторонней конференции). Данная рекомендация является альтернативой протокола SIP.

H.248 (MEGACO) — протокол, используемый между элементами телекоммуникационных сетей: шлюзом (Media Gateway) и контроллером шлюзов (Media Gateway Controller). Поддерживает различные системы сигнализации сетей с коммутацией каналов, включая тоновую сигнализацию, ISDN, ISUP, QSIG и GSM. Закреплен как стандартный протокол IMS, наряду с SIP и Diameter. Особенностью протокола является то, что каждое сообщение является транспортным механизмом передачи команд, а не самой командой, в отличие от большинства прочих телекоммуникационных протоколов.

Голосовые кодеки (codec, от coder/decoder — шифратор/дешифратор) — устройство либо программное обеспечение для преобразования речевого сигнала. Существуют различные алгоритмы сжатия, которые отличаются скоростью цифрового потока и типом лицензии.

2.2. Назначение

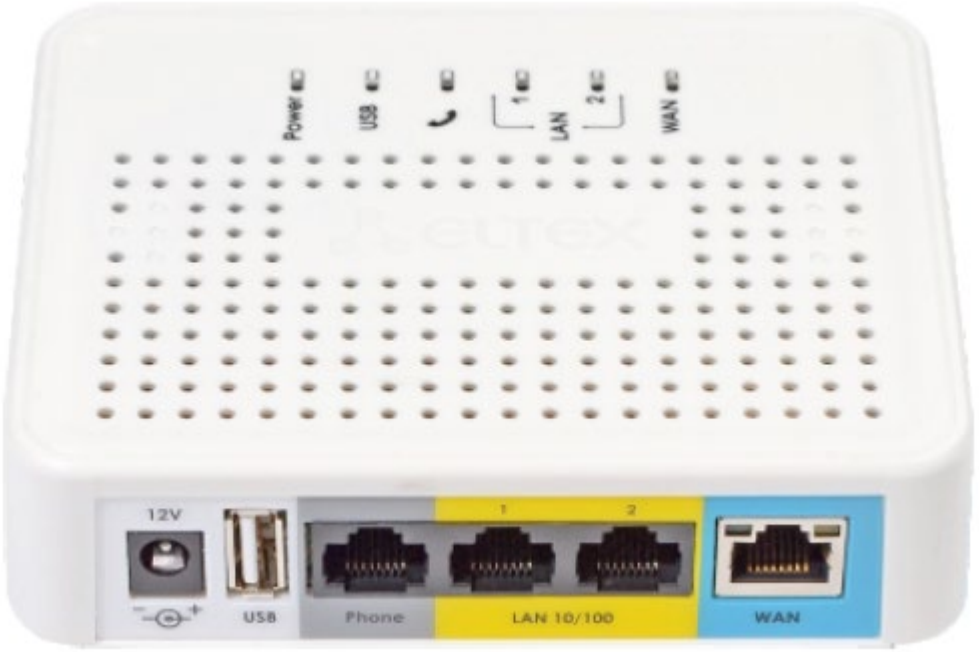


Рисунок 1. Внешний вид *TAU-1M.IP*

Устройство *TAU-1M.IP* – высокопроизводительный абонентский шлюз IP-телефонии с полным набором функций, позволяющих потребителю использовать преимущества IP-телефонии.

Абонентский шлюз *TAU-1M.IP* предназначен для подключения аналогового телефонного аппарата или факс-модема к IP-сети. Благодаря встроенному маршрутизатору устройство обеспечивает возможность подключения оборудования локальной сети к сети широкополосного

доступа. К устройству можно подключить до двух компьютеров, доступ в интернет для которых возможен с помощью встроенных функций NAT/DHCP-сервера. USB-разъем используется для подключения внешнего накопителя, 3G/4G USB-модема или Wi-Fi-адаптера.

2.3. Характеристики устройства

Интерфейсы:

- FXS: 1 порт RJ-11;
- LAN: 2 порта Ethernet RJ-45 10/100BASE-T;
- WAN: 1 порт Ethernet RJ-45 10/100BASE-T;
- USB: 1 порт USB2.0.

Функции:

- сетевые функции:
  - работа в режиме «моста» или «маршрутизатора»;
  - поддержка PPPoE (PAP, SPAP и CHAP авторизация, PPPoE компрессия);
  - поддержка PPTP;
  - поддержка L2TP;
  - поддержка статического адреса и DHCP (DHCP-клиент на стороне WAN, DHCP-сервер на стороне LAN);
  - поддержка DNS;
  - поддержка NAT;
  - сетевой экран;
  - поддержка NTP;
  - поддержка механизмов качества обслуживания QoS (QoS по DSCP и 802.1P);
- поддержка функций IPTV;
- протоколы IP-телефонии: SIP;
- эхо компенсация (рекомендации G.168);
- детектор активности речи (VAD);
- генератор комфортного шума;
- обнаружение и генерирование сигналов DTMF;
- передача DTMF (INBAND, rfc2833, SIP INFO);
- передача факса:
- – G.711A/G.711U;

- – T.38;
- работа с SIP-сервером и без него;
- функции ДВО:
  - – удержание вызова – Call Hold;
  - – передача вызова – Call Transfer;
  - – уведомление о поступлении нового вызова – Call Waiting;
  - – переадресация по занятости – Call Forward at Busy;
    - – переадресация по неответу – Call Forward at No answer;
    - – безусловная переадресация – Call Forward Unconditional;
    - – не беспокоить – DND;
    - – Caller ID: FSK, DTMF;
    - – горячая линия – Hotline;
  - – групповой вызов;
  - – перехват вызова – Call Pickup;
  - – трехсторонняя конференция;
  - – гибкий план нумерации;
- обновление ПО через web-интерфейс;
- поддержка DHCP-based autoprovisioning;
- TR-069;
- удаленный мониторинг, конфигурирование и настройка: Web-интерфейс, Telnet.

На рисунке 2 приведена схема включения *TAU-1M.IP*.

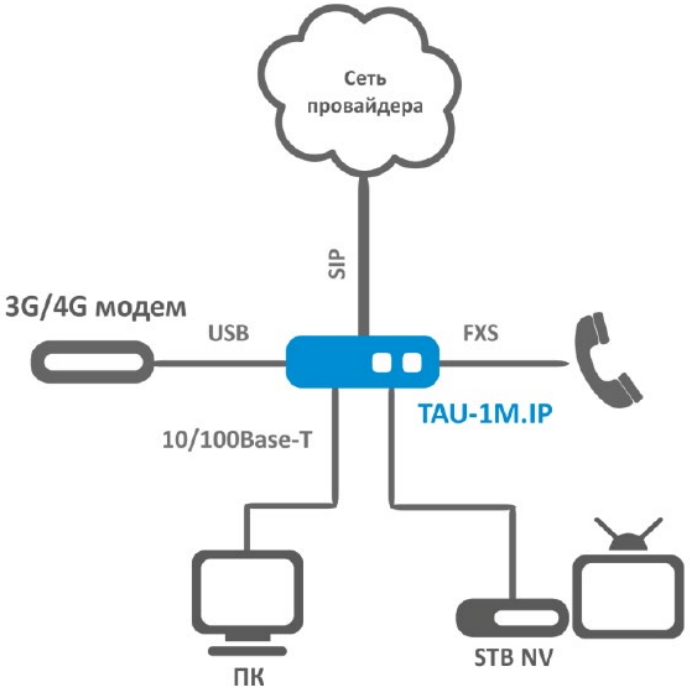


Рисунок 2 – Функциональная схема использования *TAU-1M.IP*

2.4. Конструктивное исполнение

Внешний вид верхней панели устройства TAU-1M.IP приведен на рисунке 3.

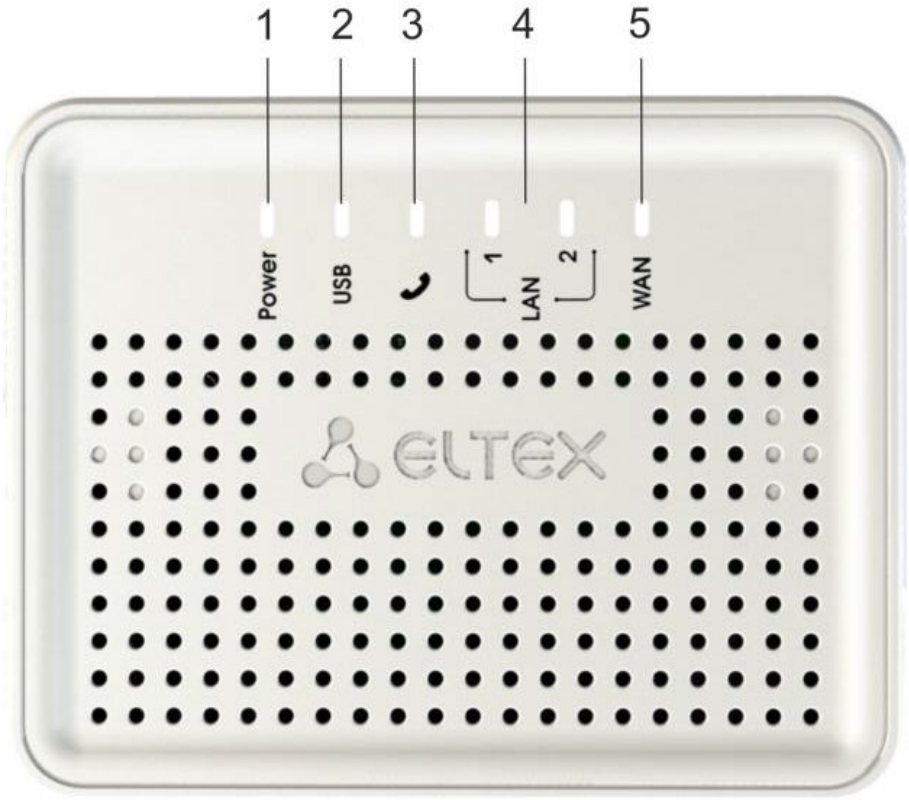


Рисунок 3 – Внешний вид верхней панели TAU-1M.IP

Таблица 1 – Описание индикаторов и органов управления передней панели

Элемент передней панели		**Описание**
1	Power	индикатор питания и статуса работы устройства
2	USB	индикатор работы внешнего USB-устройства (USB flash, внешний жесткий диск, 3G/4G USB-модем)
3		индикатор работы аналогового телефонного порта
4	LAN	индикаторы портов LAN-интерфейса
5	WAN	индикатор WAN-интерфейса

Внешний вид задней панели устройства *TAU-1M.IP* приведен на рисунке 4.

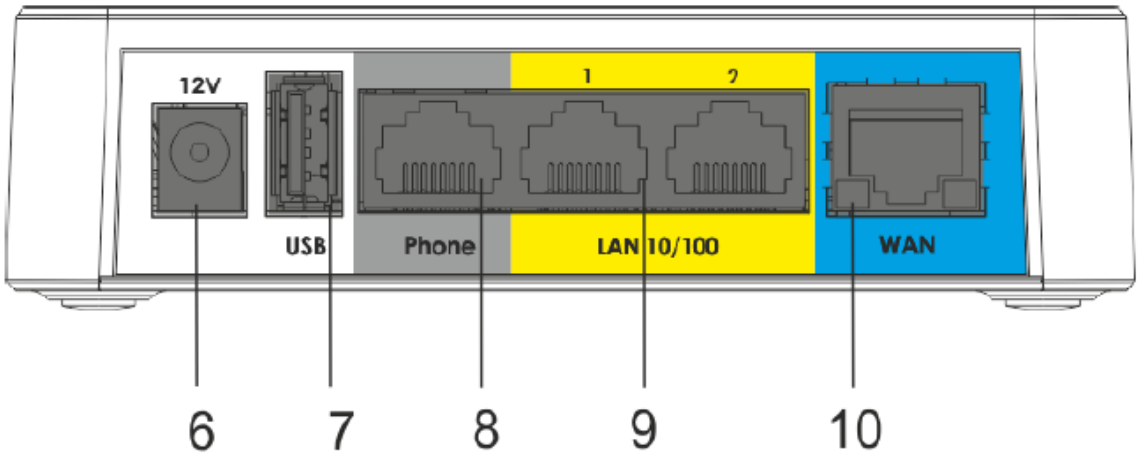


Рисунок 4 – Внешний вид задней панели *TAU-1M.IP*

На задней панели устройства *TAU-1M.IP* расположены следующие разъемы и органы управления, Таблица 3.

Таблица 2 – Описание индикаторов и органов управления задней панели *TAU-1M.IP*

Элемент задней панели		**Описание**
6	12V	разъем для подключения адаптера питания
7	USB	разъем USB для подключения внешнего USB-устройства (USB flash, жесткий диск, 3G/4G USB-модем)
8	Phone	разъем RJ-11 для подключения аналогового телефонного аппарата
9	LAN	2 порта 10/100BASE-T Ethernet (разъем RJ-45) для подключения локальных сетевых устройств
10	WAN	порт 10/100BASE-T (разъем RJ-45) для подключения к внешней сети

Текущее состояние устройства *TAU-1M.IP* отображается при помощи индикаторов **WAN, LAN, Phone, Power** – расположенных на верхней панели. Перечень состояний индикаторов приведен в таблице 4.

Таблица 4 – Световая индикация состояния устройства серии *TAU-1M.IP*

Индикатор	**Состояние индикатора**	**Состояние устройства**
WAN	горит (зеленым – 10 Mbps, оранжевым – 100Mbps)	установлено соединение между станционным терминалом и абонентским устройством
	мигает	процесс пакетной передачи данных по WAN-интерфейсу
LAN	горит (зеленым – 10 Mbps, оранжевым – 100Mbps	установлено соединение с подключенным сетевым устройством
	мигает	процесс пакетной передачи данных по LAN-интерфейсу
	зеленый, горит постоянно	телефонная трубка поднята (линия активна)
	не горит	трубка положена, нормальная работа
	зеленый, в течение секунды мигает с частотой 20 Гц, затем 4 секунды пауза	на телефонный порт поступает входящий вызов
	зеленый, периодическое редкое	отсутствует регистрация абонентского порта

	мигание	на SIP-прокси сервере
	зеленый, двойные короткие мигания с интервалом в 3 секунды	идет тест линии
USB	зеленый, горит	USB-устройство подключено
	не горит	USB-устройство не подключено
Power	зеленый, горит постоянно	включено питание устройства, нормальная работа
	оранжевый, горит постоянно	отсутствует выход в Интернет
	красный, горит постоянно	загрузка устройства
	периодическое попеременное мигание красным и зеленым цветами	сброс устройства к заводским настройкам

2.5. Сброс к заводским настройкам

Для запуска устройства с заводскими настройками необходимо в загруженном состоянии нажать и удерживать кнопку «F», которая находится на боковой панели устройства, пока индикатор «Power» начнет периодически мигать попеременно красным и зеленым цветами. Произойдет автоматическая перезагрузка устройства. При заводских установках на WAN-интерфейсе запущен DHCP-клиент, адрес интерфейса LAN - *192.168.1.1,* маска подсети – *255.255.255.0*; имя пользователя/пароль для доступа через web-интерфейс: admin/password.

2.6. План нумерации

План нумерации задается при помощи регулярных выражений в поле «Настройка плана нумерации».

Ниже приводится структура и формат регулярных выражений, обеспечивающих различные возможности набора номера.

Структура регулярного выражения:

Sxx, Lxx ( ),

где

xx – произвольные значения таймеров S и L;

() – границы плана нумерации.

Основой являются обозначения для записи последовательности набранных цифр. Последовательность цифр записывается с помощью нескольких обозначений: цифры, набираемые с клавиатуры телефона: 0, 1, 2, 3, …, 9, # и *. **Использование символа # в диалплане может блокировать завершение набора с помощью этой клавиши!**

Последовательность цифр, заключённая в квадратные скобки, соответствует любому из заключённых в скобки символу.

– Пример: ([1239]) – соответствует любой из цифр 1, 2, 3 или 9.

Через тире может быть указан диапазон символов. Чаще всего используется внутри квадратных скобок.

– Пример 1: (1-5) – любая цифра от 1 до 5.

– Пример 2:([1-39]) – пример из предыдущего пункта с иной формой записи.

Символ X соответствует любой цифре от 0 до 9.

– Пример: (1XX) - любой трёхзначный номер, начинающийся на 1.

«.» - повторение предыдущего символа от 0 до бесконечности раз.

«+» – повторение предыдущего символа от 1 до бесконечности раз.

{a,b} – повторение предыдущего символа от a до b раз;

{a,} – повторение предыдущего символа не меньше a раз;

{,b} – повторение предыдущего символа не больше b раз.

– Пример: (810X.) – международный номер с любым количеством цифр.

Настройки, влияющие на обработку диалплана:

– Interdigit Long Timer (буква «L» в записи плана нумерации) – время ожидания ввода следующей цифры в том случае, если нет шаблонов, подходящих под набранную комбинацию;

– Interdigit Short Timer (буква «S» в записи плана нумерации) – время ожидания ввода следующей цифры, если с набранной комбинацией полностью совпадает хотя бы один шаблон и при этом имеется еще хотя бы один шаблон, до полного совпадения с которым необходимо осуществить донабор номера.

Дополнительные возможности:

1. Замена набранной последовательности

Синтаксис: *<arg1:arg2>*

Данная возможность позволяет заменить набранную последовательность на любую последовательность набираемых символов. При этом второй аргумент должен быть указан определённым значением, оба аргумента могут быть пустыми.

– Пример: (<83812:> XXXXXX) - данная запись будет соответствовать набранным цифрам 83812, но эта последовательность будет опущена и не будет передана на SIP-сервер.

2. Вставка тона в набор

При выходе на межгород (в офисных станциях - на город) привычно слышать ответ станции, что можно реализовать вставкой запятой в нужную позицию последовательности цифр.

– Пример: (8, 770) - при наборе номера 8770 после цифры 8 будет выдан непрерывный тон.

3. Запрет набора номера

Если в конце шаблона номера добавить восклицательный знак ‘!’, то набор номеров, соответствующих шаблону, будет заблокирован.

– Пример: (8 10X xxxxxxx ! | 8 xxx xxxxxxx) – выражение разрешает набор только междугородних номеров и исключает международные вызовы.

4. Замена значений таймеров набора номера

Значения таймеров могут быть назначены как для всего диалплана, так и для определённого шаблона. Буква «S» отвечает за установку *«Interdigit Short Timer»*, а «L» - за *«Interdigit Long Timer»*. Значения таймеров может быть указано для всех шаблонов в диалплане, если значения перечислены до открывающейся круглой скобки.

– Пример: S4 (8XXX.) или S4,L8 (XXX)

Если эти значения указаны только в одной из последовательностей, то действуют только для неё. Также в этом случае не надо ставить двоеточие между ключом и значением таймаута, значение может быть расположено в любом месте шаблона.

Пример: (S4 8XXX. | XXX) или ([1-5] XX S0) – запись вызовет мгновенную передачу вызова при наборе трехзначного номера, начинающегося на 1,2, … , 5.

5. Набор по прямому адресу (IP Dialing)

Символ «@», поставленный после номера, означает, что далее будет указан адрес сервера, на который будет отправлен вызов на набранный номер. Рекомендуется использовать *«IP Dialing»*, а также приём и передачу

вызовов без регистрации *(«Call Without Reg», «Answer Without Reg»*). Это может помочь в случае отказа сервера.

Кроме того, формат адреса с IP Dialing может быть использован в номерах, предназначенных для переадресации звонков.

– Пример 1: (8 xxx xxxxxxx) - 11-значный номер, начинающийся на 8.

– Пример 2: (8 xxx xxxxxxx | <:8495> xxxxxxx) - 11-значный номер, начинающийся на 8, если введён 7-ми значный, то добавить к передаваемому номеру 8495.

– Пример 3: (0[123] | 8 [2-9]xx [2-9]xxxxxx) - набор номеров экстренных служб, а так же некоторого странного набора междугородних номеров.

– Пример 4: (S0 <:82125551234>) - быстрый набор указанного номера, аналог режима «Hotline» на других шлюзах.

– Пример 5: (S5 <:1000> | xxxx) - данный диалплан позволяет набрать любой номер, состоящий из цифр, а если ничего не введено в течение 5 секунд, вызвать номер 1000 (допустим, это секретарь).

– Пример 6: (*5x*xxxx*x#|*2x*xxxxxxxxxxx#|#xx#|[2-7]xxxxx|8, [2-9]xxxxxxxxx|8, 10x.|1xx<:@10.110.60.51:5060>).

– Пример 7: (1xx|0[1-9]|00[1-8]|*5x*xxxx*x#|*2x*xxxxxxxxxxx#|#xx#|[2-7]xxxxx|8, [2-9]xxxxxxxxx|8, 10x.).

Установка профилей dialplan

Для каждого направления можно выбрать не более одного профиля dialplan, который будет определять параметры вызовов на этом направлении. Настройки профилей производятся в разделе «Профили dialplan». Для каждого направления настройка альтернативного профиля указывается в круглых скобках после слова «profile:».

Пример: **Пример:([23]xxx(profile:0)**

3. Порядок выполнения лабораторной работы

3.1. Описание работы

Целью лабораторной работы является изучение способов конфигурирования устройства, сброс устройства на заводские настройки, работа с файлами конфигурации, ознакомление с основными настройками TAU.1M-IP включая параметры SIP протокола, особенности настройки абонентских портов и других функций.

3.2. Порядок выполнения работы

Для выполнения работы необходимо выполнить следующие действия:

1) Определить устройство и его IP-адрес с которым будет проводится работа. Устройства распределяются следующим образом:

Вариант 1 (группа 1) — TAU#1 - 192.168.2.11

Вариант 2 (группа 2) — TAU#2 - 192.168.2.12

Вариант 3 (группа 3) — TAU#3 - 192.168.2.13

Вариант 4 (группа 4) — TAU#4 - 192.168.2.14

Вариант 5 (группа 5) — TAU#5 - 192.168.2.15

Вариант 6 (группа 6) — TAU#6 - 192.168.2.16

Установить следующие параметры сети:

1. Режим работы: Мост
2. Протокол: Static
3. IP по варианту
4. Маска подсети 255.255.255.0
5. Шлюз по умолчанию 192.168.2.2

В случае если к указанному устройства нет доступа (неизвестный IP-адрес, логин/пароль) то необходимо выполнить сброс настроек устройства. Для дальнейшей работы нужно установить адрес согласно варианту.

2) Подключиться к web-конфигуратору TAU при помощи любого веб-браузера (напр. Google Chrome). Для этого необходимо в адресной строке браузера указать необходимый IP-адрес устройства, порт для подключения по протоколу HTTP (80) и HTTPS (443) стандартный, т.е. дополнительно его указывать не нужно. Логин: admin, пароль: password;

3) Изучить расположение всех вкладок и их содержимое рекурсивно (с учетом всех вложенных вкладок);

4) Перейти в раздел «Сервисные функции» => «Управление конфигурацией». Создать резервную копию текущей конфигурации в формате *.tar и сохранить на локальный диск;

5) Внести изменения в конфигурацию устройства, применить и сохранить изменения. После чего использовать функцию восстановления конфигурации из файла. Просмотреть измененные параметры конфигурации;

6) Определить план нумерации по схеме сети по прямому адресу:

Вариант 1 (группа 1) — 201

Вариант 2 (группа 2) — 202

Вариант 3 (группа 3) — 203

Вариант 4 (группа 4) — 204

Вариант 5 (группа 5) — 205

Вариант 6 (группа 6) — 206

Символ «@», поставленный после номера, означает, что далее будет указан адрес сервера, на который будет отправлен вызов на набранный номер. Рекомендуется использовать *«IP Dialing»*, а также приём и передачу вызовов без регистрации («Call Without Reg», «Answer Without Reg»). Это может помочь в случае отказа сервера.

Кроме того, формат адреса с *IP Dialing* может быть использован в номерах, предназначенных для переадресации звонков.

Для выхода на абонентов встречного TAU без использования сервера регистрации можно использовать следующее правило *number@host*, где *host* — ip-адрес встречного TAU, например:

21000xx@10.10.2.222

Последовательно позвонить всем абонентам сети.

7) Изменить план нумерации, установив запрет исходящего вызова на номер соседнего абонента. Проверить работоспособность.

8) Выполнить регистрацию абонента на SIP-сервере с IP-адресом: 192.168.2.200. Параметры аутентификации: имя и пароль пользователя совпадают с номером телефона. SIP домен совпадает с номером телефона.

Определить план нумерации по схеме сети с использованием сервера регистрации.

8) Перейти во вкладку «Мониторинг» и проверить состояние регистрации абонентских портов.

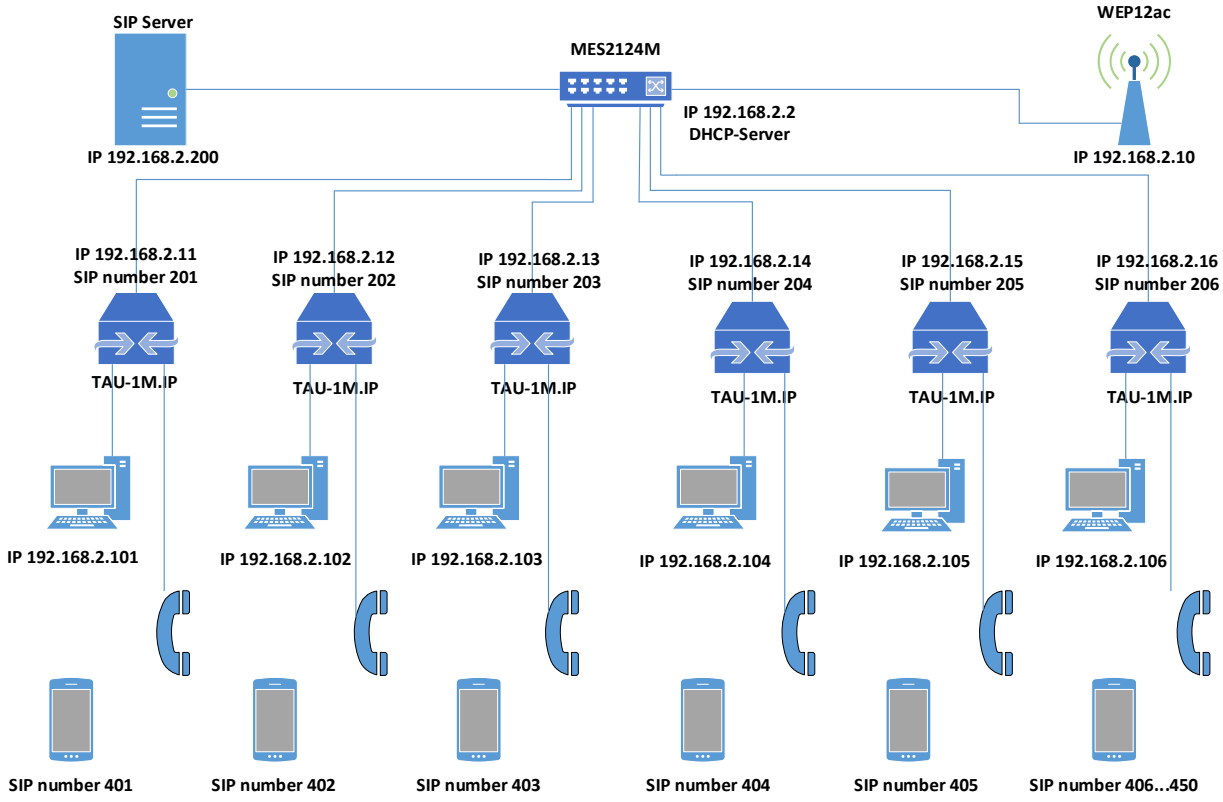
9) В случае успешной регистрации необходимо поочередно установить соединение между всеми доступными элементами сети доступными через sip-сервер;

10) Используя встроенные средства тестирования и статистики провести тестирование абонентских линий, проанализировать результат, а также получить FXS статистику порта;

11) Подключить смартфон к WI-FI сети **Eltex RPU VAP** пароль: 33333333. Установить на смартфон приложение для SIP телефонии. Выбрать из диапазона 401 – 450 номер телефона. Выполнить регистрацию абонента на SIP-сервере с IP-адресом: 192.168.2.200. Параметры аутентификации: имя и пароль пользователя совпадают с номером телефона. SIP домен совпадает с номером телефона.

12) В случае успешной регистрации необходимо поочередно установить соединение между всеми доступными элементами сети через sip-сервер.

Схема сети к лаб. работе



4. Конфигурирование Ethernet коммутаторов MES

Цели и задачи лабораторной работы

Целью лабораторной работы является изучение базовых принципов управления и конфигурирования *Ethernet*-коммутаторов доступа серии *MES2000*, их технических характеристик и некоторых схем использования данных устройств в сети.

4. Краткие теоретические сведения

4.1. Назначение коммутатора

Сетевым коммутатором (или *свитчем* – от англ. *switch* – переключатель) называют устройство, предназначенное для соединения нескольких узлов компьютерной сети в пределах одного или нескольких сегментов сети. В отличие от маршрутизатора (англ. *router*) коммутатор является устройством **канального** (второго) уровня модели *OSI* (англ. *open systems interconnection basic reference model* – **Б**азовая **Э**талонная **М**одель **В**заимодействия **О**ткрытых **С**истем (ЭМВОС)), что и определяет главные различия между ними. Коммутатор не занимается расчетом маршрута для дальнейшей передачи пакетов по сети, анализируя различные факторы, как это делает маршрутизатор. Коммутатор только передает данные от одного порта к другому на основе содержащейся в пакете информации. Обычно признаком выбора выходного порта служит *MAC*-адрес[3] устройства, к которому передаются данные. В свою очередь коммутатор в отличие от концентратора (англ. *hub*) или повторителя (англ. *repeater*) не просто транслирует порты ко всем выходам, которые у него есть, а к одному, заранее выбранному.

4.2. Коммутаторы доступа серии MES2000

Коммутатор доступа осуществляет подключение конечных пользователей к сети крупных предприятий, предприятий малого и среднего бизнеса и к сетям операторов связи с помощью интерфейсов *Fast* и *Gigabit Ethernet*.

Данная лабораторная работа выполняется с использованием оборудования одного из ведущих российских вендоров телекоммуникационного оборудования – ООО «Предприятие «ЭЛТЕКС». В лабораторной работе задействованы коммутаторы доступа **MES2124M** и **MES2324P.** Сетевые коммутаторы **MES2124M** и **MES2324P** имеют в своём

[3] ***MAC*-адрес** (англ. *Media Access Control* — управление доступом к среде) – уникальный идентификатор, присваиваемый каждой единице активного оборудования или некоторым их интерфейсам в компьютерных сетях *Ethernet*.

составе 24 порта *Gigabit Ethernet* с электрическими интерфейсами (и поддержкой *PoE+* (англ. *Power over Ethernet* – технология, позволяющая передавать удалённому устройству электрическую энергию вместе с данными через стандартную витую пару в сети *Ethernet*) для **MES2324P**) и 4 порта *Gigabit Ethernet*, совмещенные со слотами для установки *SFP*-трансиверов (англ. *Small Form-factor Pluggable* – промышленный стандарт модульных компактных приёмопередатчиков, используемых для присоединения платы сетевого устройства к оптическому волокну или неэкранированной витой паре).

4.3. Основные технические характеристики

Таблица 2.3.1. Основные технические параметры используемых коммутаторов

Параметры	**MES2124M**	**MES2324P**
Общие параметры		
Пакетный процессор	Marvell 98DX1035 / 98DX3035	Marvell 98DX3236-A1 (AlleyCat3)
Интерфейсы	24x10/100/1000Base-T 4x (10/100/1000Base-T / 1000Base-X Combo)	24x10/100/1000Base-T (PoE+) 4x(10G Base-X(SFP+)/1000Base-X (SFP))
Пропускная способность	56 Гбит/с	128 Гбит/с
Объем буферной памяти	8 Мбит	12 Мбит
Таблица MAC-адресов	16К записей (часть MAC-адресов резервируется для использования системой)	16К
Поддержка VLAN	до 4К	до 4К
Скорость передачи данных	Электрические интерфейсы 10/100/1000 Мбит/с. Оптические интерфейсы 1Гбит/с	Оптические интерфейсы 1/10 Гбит/с Электрические интерфейсы 10/100/1000 Мбит/с
Управление		
Локальное управление	Консоль RS-232	

Удаленное управление	SNMP, Telnet, SSH, WEB	
Физические характеристики и условия окружающей среды		
Источник питания	сеть переменного тока 110-250В, 50 Гц или сеть постоянного тока 48В. Потребляемая мощность: - не более 30 Вт.	сеть переменного тока: 220В±20%, 50 Гц Потребляемая мощность: - не более 1600 Вт.
Габаритные размеры	430x44x180 мм	440 x 203 x 44 мм
Интервал рабочих температур	от -10 до +45 ^{o}C	от -20 до +45 ^{o}C

4.4. Конструктивное исполнение

Коммутаторы выполнены в металлическом корпусе с возможностью установки в 19” каркас типоразмера *1U*.

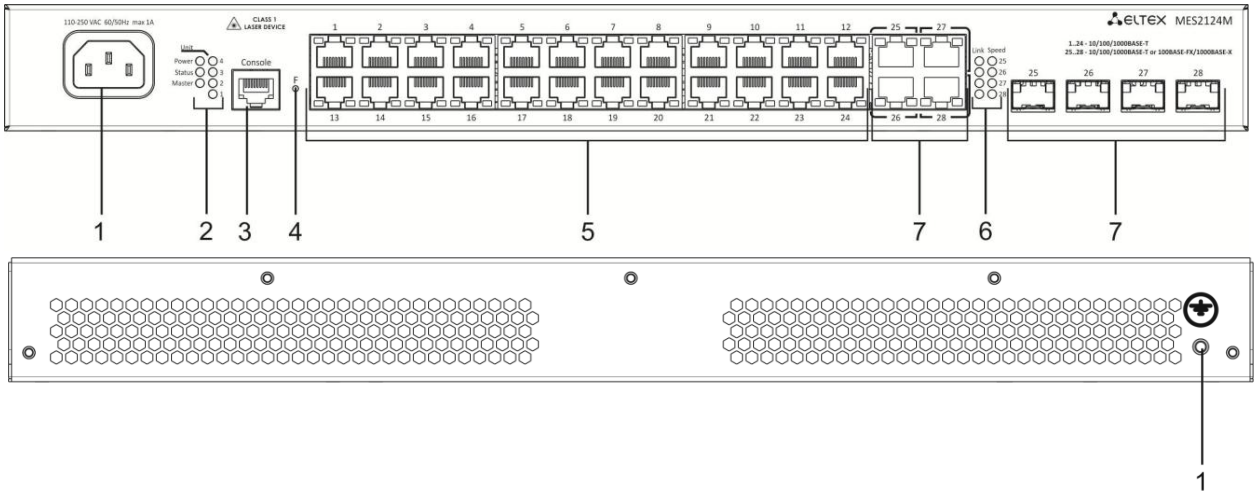


Рисунок 2.4.1. Внешний вид передней и задней панелей MES2124M

Таблица 2.4.1. Описание разъемов, индикаторов и органов управления передней панели MES2124M

№	Элемент передней панели	Описание
1	110-250VAC, 60/50Hz max 1A	Разъем для подключения к источнику электропитания переменного тока
	Power	Индикатор питания устройства
	Status	Индикатор состояния устройства

2	Master	Индикатор режима работы устройства в стеке - ведущий или ведомый
	Unit ID (1-4)	Индикаторы номера устройства в стеке
3	Console	Консольный порт RS-232 для локального управления устройством
4	F	Функциональная кнопка для перезагрузки устройства и сброса к заводским настройкам: - при нажатии на кнопку длительностью менее 10 с. происходит перезагрузка устройства; - при нажатии на кнопку длительностью более 10 с. происходит сброс настроек устройства до заводской конфигурации.
5	[1 .. 24]	24 порта 10/100/1000 Base-T (RJ-45)
6	Link/Speed	Световая индикация состояния оптических интерфейсов
7	25, 26, 27, 28	Комбо-порты: порты 10/100/1000 Base-T (RJ45) и слоты для установки трансиверов 1000Base-FX/1000Base-X Combo

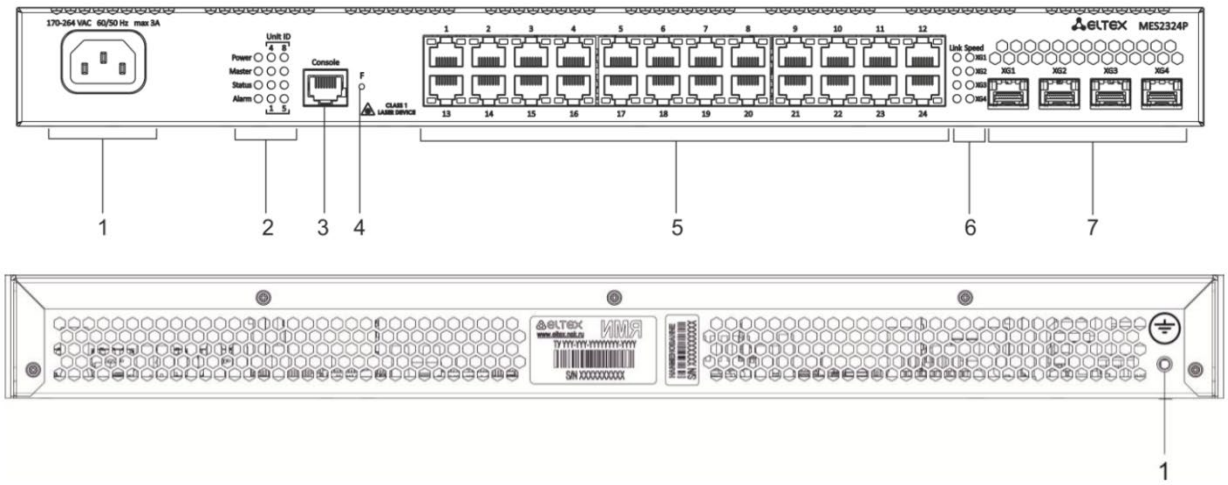


Рисунок 2.4.2. Внешний вид передней и задней панелей MES2324P

Таблица 2.4.1. Описание разъемов, индикаторов и органов управления передней панели MES2324P

№	Элемент передней панели	Описание
1	110-250VAC, 60/50Hz max 2A	Разъем для подключения к источнику электропитания переменного тока
2	UnitID	Индикаторы номера устройства в стеке
	Power	Индикатор питания устройства
	Master	Индикатор режима работы устройства в стеке - ведущий или ведомый
	Status	Индикатор состояния устройства
3	Console	Консольный порт RS-232 для локального управления устройством
4	F	Функциональная кнопка для перезагрузки устройства и сброса к заводским настройкам: - при нажатии на кнопку длительностью менее 10 с. происходит перезагрузка устройства; - при нажатии на кнопку длительностью более 10 с. происходит сброс настроек устройства до заводской конфигурации.
5	[1 .. 24]	24 порта 10/100/1000 Base-T (RJ-45)
6	Link/Speed	Световая индикация состояния оптических интерфейсов
7	XG1, XG2, XG3, XG4,	Слоты для установки трансиверов 10GSFP+/1GSFP

На задней панели устройств расположен болт для заземления устройства, который обозначен символом ⏚ (1).

С подробным описанием функционала и настроек коммутаторов можно ознакомиться в руководстве по эксплуатации.

4.5. Управление устройством. Режимы конфигурирования

Для конфигурирования настроек коммутатора используется четыре основных режима. В каждом режиме доступен определенный список команд.

Внимание! Ввод символа «?» служит для просмотра набора команд, доступных в каждом из режимов.

Командный режим (EXEC), данный режим доступен сразу после успешной загрузки коммутатора и ввода имени пользователя. Приглашение системы в этом режиме состоит из имени устройства (host name) и символа “>”.

```
console>
```

Если имя устройства не назначено, то вместо него используется слово “console”.

Привилегированный командный режим (privileged EXEC), этот режим доступен при входе привилегированного пользователя. Вход в режим должен быть обязательно защищен паролем. Только в привилегированном режиме доступны команды изменения системных параметров коммутатора. В привилегированном режиме в строке приглашения системы используется символ «#». Для перехода из режима EXEC в привилегированный режим может быть использована команда ***enable***.

```
console> enable
enter password:
console#
```

Режим глобального конфигурирования (global configuration), данный режим предназначен для задания общих настроек коммутатора. Команды режима глобальной конфигурации доступны из любого подрежима конфигурации. Вход в режим осуществляется командой ***configure***.

```
console# configure
console(config)#
```

Режим конфигурирования интерфейса (interface configuration), данный режим предназначен для конфигурирования интерфейсов (порт, группа портов, интерфейс VLAN) коммутатора. Вход в режим осуществляется из режима глобального конфигурирования, для каждого

интерфейса своей командой (в примере ниже команда для входа в режим конфигурирования интерфейса VLAN с VID=1).

```
console(config)# interface vlan 1
console (config-if)#
```

5. Порядок выполнения лабораторной работы

Внимание! Перед выполнением работы необходимо сбросить устройство к заводским настройкам.

5.1. Практическое задание №1. Базовое конфигурирование MES

1) Выполнить подключение персонального компьютера к консольному порту коммутатора, указанного преподавателем. Данный порт предоставляет доступ к диагностике, управлению и мониторингу устройства.

2) Запустить *minicom*[4] от пользователя *root* с помощью следующей команды:

```
$ sudo minicom
```

Ввести пароль от учетной записи. После регистрации на устройстве в консоли появится системное приглашение интерфейса командной строки CLI (англ. *Command line interface*).

```
console>
```

3) Создать учетную запись с уровнем привилегий 1. Настроить пароль для перехода в привилегированный режим с уровнем 15.

Внимание! Уровень привилегий 1 разрешает доступ к устройству, но запрещает настройку. Уровень привилегий 15 разрешает как доступ, так и настройку устройства.

Для создания нового пользователя системы или настройки любого из параметров – имени пользователя, пароля, уровня привилегий, используются команды:

```
console# configure
```

[4] **Minicom** – программа для использования последовательного порта (COM) в операционных системах Linux.

```
console(config)# username name password password
privilege {1-15}
```

Пример команд для задания пользователю «**admin**» пароля «**eltex**» и создания пользователя «**operator**» с паролем «**pass**» и уровнем привилегий 1:

```
console# configure
console(config)# username admin password eltex
console(config)# username operator password pass
privilege 1
console(config)# exit
console#
```

4) Создать учетную запись с уровнем привилегий 15.

Внимание! Просмотреть локальную базу данных пользователей и их привилегий можно командой:

```
console# show users accounts
```

5) Задать Hostname устройству согласно его наименованию.

Для задания сетевого имени устройства используются команды:

```
console# configure
console(config)# hostname name
```

Вернуть сетевое имя устройства в значение по умолчанию:

```
console(config)# no hostname
```

6) Задать сетевые настройки для доступа к устройству по IP.

Для возможности управления коммутатором из сети необходимо назначить устройству IP-адрес, маску подсети и, в случае управления из другой сети, шлюз по умолчанию. IP-адрес можно назначить любому интерфейсу – VLAN[5], физическому порту, группе портов (по умолчанию на интерфейсе VLAN 1 назначен IP-адрес 192.168.1.239, маска 255.255.255.0).

[5] **VLAN** (англ. *Virtual Local Area Network* - логическая («виртуальная») локальная компьютерная сеть) - группа устройств, имеющих возможность взаимодействовать между собой напрямую на канальном уровне, хотя физически при этом они могут быть подключены к разным сетевым коммутаторам. В современных сетях VLAN — главный механизм для создания логической топологии сети, не зависящей от её физической топологии.

IP-адрес шлюза должен принадлежать к той же подсети, что и один из IP-интерфейсов устройства.

Настроить интерфейс VLAN 1 коммутатора согласно варианту:

- IP-адрес

Варианты 1,2 (Группа 1) – MES2124M#1 - *192.168.1.11*

Варианты 3,4 (Группа 2) – MES2124M#2 - *192.168.1.12*

Варианты 5,6 (Группа 3) – MES2324P#3 - *192.168.1.13*

- Маска подсети – *255.255.255.0*
- IP-адрес шлюза по умолчанию – *192.168.1.1*

Пример команд настройки IP-адреса для интерфейса VLAN 1.

IP-адрес, назначаемый для интерфейса VLAN 1 – 192.168.16.144
Маска подсети – 255.255.255.0
IP-адрес шлюза по умолчанию – 192.168.16.1

```
console# configure
console(config)# interface vlan 1
console(config-if)# ip address 192.168.16.144 /24
console(config-if)# exit
console(config)# ip default-gateway 192.168.16.1
console(config)# exit
console#
```

Для того чтобы убедиться, что адрес был назначен интерфейсу, введите команду:

```
console# show ip interface vlan 1
```

IP Address	I/F	I/F Status admin/oper	Type	Directed Broadcast	Prec	Redirect	Status
192.168.16.144/24	vlan 1	UP/DOWN	Static	disable	No	enable	Valid

7) Выполнить подключение к коммутатору по протоколу Telnet[6] (или протоколу SSH[7]). Для этого соединить один из портов *Gigabit Ethernet*

[6] **Telnet** (сокр. от англ. ***tele****type* ***net****work*) – сетевой протокол прикладного уровня модели *OSI* для реализации текстового интерфейса при помощи транспорта *TCP*, а также утилита, предназначенная для создания интерактивного соединения между удаленными устройствами.

[7] **SSH** (сокр. от англ. **S**ecure **Sh**ell — «безопасная оболочка») – сетевой протокол прикладного уровня модели *OSI*, позволяющий производить удалённое управление операционной системой и туннелирование *TCP*-

коммутатора с сетевой картой персонального компьютера. Выполнить открытие Telnet-сессии (SSH-сессии) соответствующими командами:

```
$ telnet A.B.C.D
$ ssh A.B.C.D
```

где A.B.C.D – IP-адрес коммутатора. Ввести данные учетной записи.

Внимание! При использовании протокола SSH требуется разрешить на коммутаторе удаленное конфигурирование через SSH. Для этого, используя консольное соединение, запустите на коммутаторе SSH сервер командой:

```
console# configure
console(config)# ip ssh server
```

8) Просмотреть файл текущей конфигурации коммутатора. Сохранить в отчет!

```
console# show running-config
```

Внимание! При перезагрузке устройства все несохраненные данные будут утеряны. Для сохранения любых внесенных изменений в настройку коммутатора используется следующая команда:

```
console# write
```

9) Выгрузить конфигурацию коммутатора на TFTP[8]-сервер. IP-адрес TFTP-сервера: *192.168.1.5*. Название файла конфигурации должно включать модель коммутатора и номер группы.

Пример команд копирования файла текущей конфигурации коммутатора на TFTP-сервер и последующей загрузки:

```
console#copy running-config tftp://10.10.10.1/mstp.conf
console#copy tftp://10.10.10.1/mstp.conf running-config
```

Внимание! Перед выполнением данного пункта убедитесь в наличие соединения с TFTP-сервером, выполним в терминале команду **ping**.

10) Загрузить конфигурацию коммутатора с TFTP-сервера.

соединений (например, для передачи файлов). Схож по функциональности с протоколами *Telnet* и *rlogin*, но, в отличие от них, шифрует весь трафик, включая и передаваемые пароли.

[8] **TFTP** (англ. ***T**rivial **F**ile **T**ransfer **P**rotocol* – простой протокол передачи файлов) используется для загрузки бездисковых рабочих станций, загрузки обновлений и конфигураций в «умные» сетевые устройства. TFTP, в отличие от FTP, не содержит возможностей аутентификации (хотя возможна фильтрация по IP-адресу) и основан на транспортном протоколе UDP.

5.2. Практическое задание №2. Конфигурирование Ethernet портов

1) Создать VLAN с номерами согласно вариантам:

Вариант 1 (Группа 1) – VLAN 101, 201, 301, 501, 1501, 2001, 3001;

Вариант 2 (Группа 1) – VLAN 102, 202, 302, 502, 1502, 2002, 3002;

Вариант 3 (Группа 2) – VLAN 103, 203, 303, 503, 1503, 2003, 3003;

Вариант 4 (Группа 2) – VLAN 104, 204, 304, 504, 1504, 2004, 3004;

Вариант 5 (Группа 3) – VLAN 105, 205, 305, 505, 1505, 2005, 3005;

Вариант 6 (Группа 3) – VLAN 106, 206, 306, 506, 1506, 2006, 3006.

Для перехода в режим конфигурирования VLAN следует ввести следующие команды:

```
console# configure
console(config)# vlan database
console(config-vlan)#
```

Команда добавления VLAN или нескольких VLAN:

```
console(config-vlan)# vlan {VLANlist}
```

где *VLANlist*: (2…4094) – список номеров VLAN.

Команда удаления VLAN или нескольких VLAN:

```
console(config-vlan)# no vlan {VLANlist}
```

После выполнения пункта отобразить информацию о всех доступных VLAN с помощью команды:

```
console(config)# show vlan
```

Занести данные в отчет.

2) Перейти в режим конфигурирования интерфейса Ethernet. Данный режим доступен из режима конфигурирования и предназначен для задания параметров конфигурации интерфейса (порта коммутатора или группы портов, работающих в режиме разделения нагрузки), либо диапазона интерфейсов.

Порт может работать в четырех режимах:

– access – интерфейс доступа – нетегированный интерфейс для одной VLAN;

– trunk – интерфейс, принимающий только тегированный трафик, за исключением одного VLAN, который может быть добавлен с помощью команды *switchport trunk native vlan*;

– general – интерфейс с полной поддержкой 802.1q[9], принимает как тегированный, так и нетегированный трафик;

– customer – Q-in-Q интерфейс.

Настроить порты коммутатора согласно таблице 3.2.1

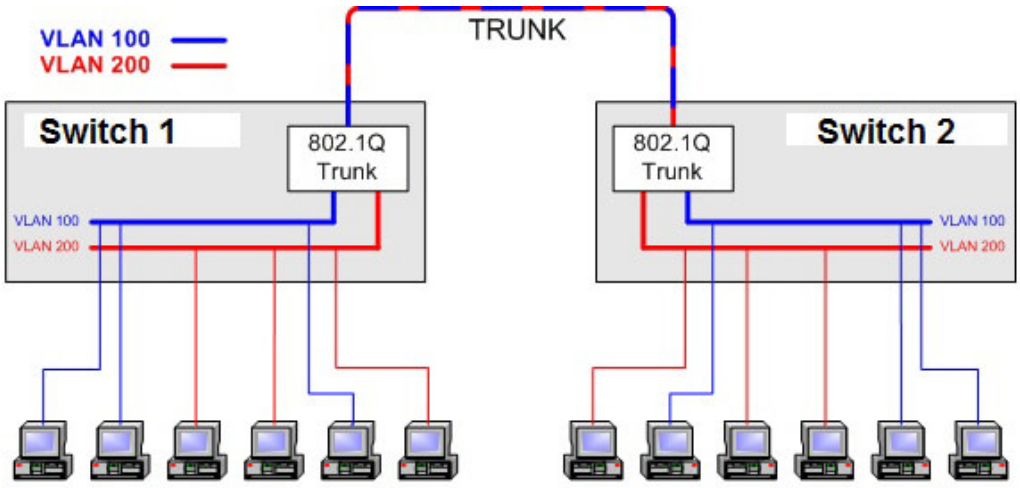
Таблица 3.2.1. Настройка портов коммутаторов

№ варианта	**Вариант 1**	**Вариант 2**	**Вариант 3**	**Вариант 4**	**Вариант 5**	**Вариант 6**
Порт	*gi1/0/2*	*gi1/0/7*	*gi1/0/2*	*gi1/0/7*	*gi1/0/2*	*gi1/0/7*
Режим	Access	Access	Access	Access	Access	Access
VLAN	101	102	103	104	105	106
Порт	*gi1/0/3*	*gi1/0/8*	*gi1/0/3*	*gi1/0/8*	*gi1/0/3*	*gi1/0/8*
Режим	Trunk	Trunk	Trunk	Trunk	Trunk	Trunk
VLAN	201	202	203	204	205	206
Native VLAN	301	302	303	304	305	306

Поясним различия между режимами портов **Access** и **Trunk**. Первый тип используется при подключении конечных хостов, таких как компьютеры, ip-телефоны, сервера и т.д. При этом указывается, в каком VLAN данный хост будет работать.

Второй предназначен в основном для подключений между коммутаторами, передавая несколько VLAN.

[9] **IEEE 802.1q** – открытый стандарт, который описывает процедуру тегирования трафика для передачи информации о принадлежности к VLAN.



Другими словами, если вы имеете более одного VLAN на транковом порту, вам необходимо указать сетевому устройству, какой из пакетов данных к какому VLAN принадлежит на другом конце соединения. Для этого и используется ***механизм тегирования*** пакетов данных с помощью VLAN тегов. VLAN тег просто вставляется в оригинальный Ethernet-кадр, добавляя необходимую информацию. Таким образом, тегированные пакеты данных содержат информацию о принадлежности к VLAN, в то время как нетегированные – нет. Типичный пример использования тегирования – это подключение между маршрутизатором и коммутатором, за которым находится несколько подключенных к нему пользователей из разных VLAN.

Для настройки одного из портов коммутатора в режим **Access** необходимо использовать следующие команды:

```
console# configure
console(config)# interface gi1/0/10
console(config-if)# switchport mode access
```

Для добавления **Access**-порта в VLAN:

```
console(config-if)# switchport access vlan 100
console(config-if)# exit
```

Для настройки одного из портов коммутатора в режим **Trunk** необходимо использовать следующие команды:

```
console# configure
console(config)# interface gi1/0/10
```

```
console(config-if)# switchport mode trunk
```

Для добавления **Trunk -**порта в VLAN:

```
console(config-if)# switchport trunk allowed vlan add
100
console(config-if)# exit
```

Для настройки заданного VLAN (например, VLAN 102) как native[10] выполните команда:

```
console(config-if)# switchport trunk native vlan 102
```

Просмотр состояния портов осуществляется командой:

```
console# show interfaces switchport GigabitEthernet
1/0/10
```

После настройки всех портов занести данную информацию в отчет. Кроме того, вновь отобразить информацию о доступных VLAN, занести в отчет.

[10] **Native VLAN** – понятие в стандарте 802.1Q, которое обозначает VLAN на коммутаторе, где все кадры идут без тэга, т.е. трафик передается нетегированным. По умолчанию это VLAN 1. Если коммутатор получает нетегированные кадры на транковом порту, он автоматически причисляет их к Native VLAN. И точно так же кадры, генерируемые с не распределенных портов, при попадании в транк-порт причисляются к Native VLAN.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Уорд. Б.* Внутреннее устройство Linux. – СПб.: Питер, 2018. – 384 с.

2. *Колисниченко Д.Н., Аллен Питер В.* LINUX: полное руководство. – СПб.: Наука и Техника, 2006. – 784 с.

3. *Cobbaut P.* Linux Fundamentals. Netsec BVBA, 2015. – 355 p.

4. Русскоязычная документация по Ubuntu. Командная строка [Электронный ресурс]. – Режим доступа: https://help.ubuntu.ru/wiki/командная_строка.

5. Русскоязычная документация по Ubuntu. Администратор в Ubuntu, или Что такое sudo [Электронный ресурс]. – Режим доступа: https://help.ubuntu.ru/wiki/суперпользователь_в_ubuntu.

6. *Колисниченко Д.Н.* Linux. От новичка к профессионалу. – СПб.: БХВ-Петербург, 2018. – 672 с.

7. *Шотсс У.* Командная строка Linux. Полное руководство. – СПб.: Питер, 2017. – 480 с.

8. Официальный сайт Vim. Documentation [Электронный ресурс]. – Режим доступа: https://www.vim.org/docs.php.

9. Официальный сайт GNU nano. Documentation [Электронный ресурс]. – Режим доступа: https://www.nano-editor.org/docs.php.

10. *Мэтью Н.* Основы программирования в Linux: Пер. с англ. / Н. Мэтью, Р. Стоунс. – СПб.: БХВ-Петербург, 2009. – 896 с.

ПРИЛОЖЕНИЯ

Приложение 1. Консольные текстовые редакторы

Vi/Vim

Первая версия текстового редактора **vi** (от англ. ***vi****sual* – экранный) написана Б. Джоем в 1976 г. Основан на более ранних строчных текстовых редакторах, в частности, **ex**. Большинство современных дистрибутивов Linux содержат не настоящий **vi**, а его улучшенную версию **vim** (сокр. от англ. *Vi* ***Im****proved* – Vi улучшенный), созданную Б. Моленаром. **vim** имеет значительное количество улучшений по сравнению с **vi** и на сегодняшний день является одним из мощнейших консольных текстовых редакторов с полной свободой настройки и автоматизации, возможными благодаря расширениям и надстройкам. Во многих системах Linux **vim** используется под символической ссылкой (или псевдонимом) **vi**, поэтому даже запуская **vi**, вы, скорее всего, запускаете **vim**.

Чтобы запустить **vi** введите команду:

```
$ vi
```

Окно терминала примет вид, показанный на рис. П1.1.

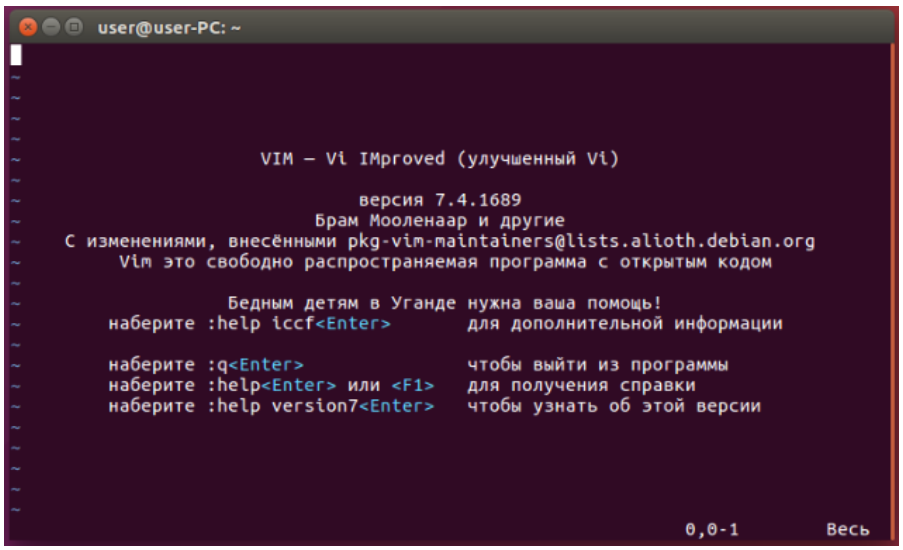


Рис. П1.1. Внешний вид терминала при запуске **vim**.

Для создания нового файла с именем `file.txt` в текущем каталоге выполним в терминале команду:

```
$ vi file.txt
```

Окно терминала при этом будет выглядеть как на рис. П1.2.

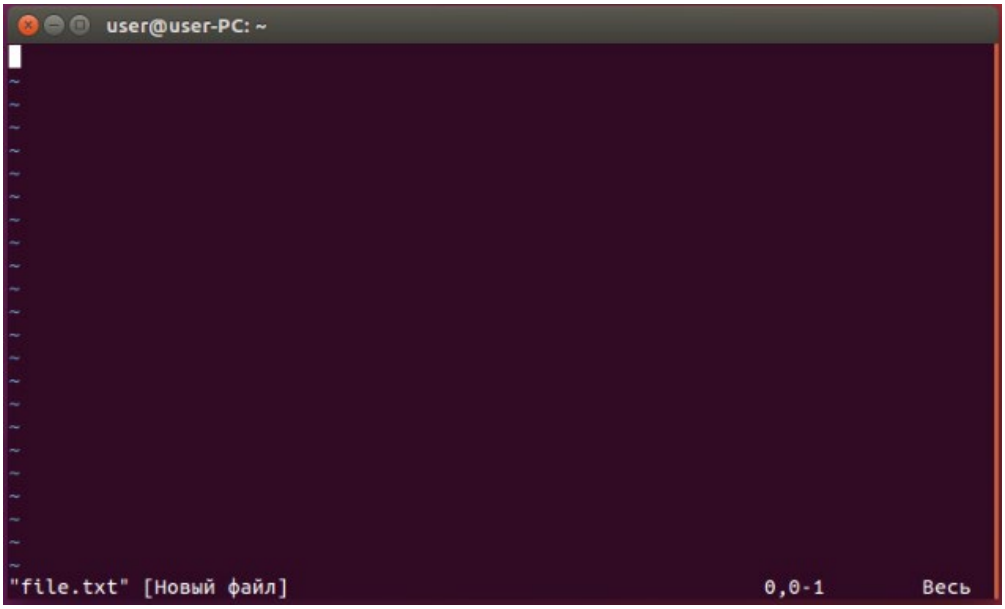


Рис. П1.2. Внешний вид терминала при создании нового файла с помощью **vim**.

Начальные символы тильды (**~**) сообщают об отсутствии текста в соответствующих строках. Для дальнейшего редактирования файла следует разобраться с принципом работы редактора.

В отличие от многих текстовых редакторов, **vim** является режимным редактором, т.е. в каждый момент времени находится в одном из двух режимов работы: *командном режиме* или *режиме вставки*. Одни и те же клавиши в разных режимах работы выполняют разные действия. По умолчанию, работа начинается в командном режиме.

Командный режим. Клавиши (в том числе алфавитно-цифровые) являются командами перемещения по тексту и редактирования. Например, клавиши **h**, **j**, **k**, **l** сдвигают курсор на одну позицию вправо, вниз, вверх, влево соответственно (также можно использовать клавиши-стрелки), клавиша **x** удаляет символ в позиции курсора и т.п. Это позволяет обойтись без использования клавиш-модификаторов («Ctrl», «Shift» и др.). Более сложные команды можно получить, комбинируя простые, например, нажатие клавиш **2dw** удаляет два слова.

Режим вставки. Алфавитно-цифровые клавиши используются для добавления текста в файл. Для перехода в режим вставки можно нажать клавишу **i** или **a**. В нижней части экрана при этом появится надпись:

```
-- ВСТАВКА --
```

Для возвращения в командный режим нажмите **«Esc»** или **«Ctrl»+C**.

Основные команды редактора **vim** приведены в Таблице П1.1.

Таблица П1.1. Основные команды **vim**

Клавиши	Команда
i **a** **o** **O**	Переход в режим вставки, вставка в позиции курсора: - перед текущим символом; - после текущего символа; - после текущей строки; - перед текущей строкой.
x	Удалить символ в позиции курсора.
dd	Удалить текущую строку.
dw	Удалить все символы от символа в текущей позиции курсора до начала следующего слова.
u	Отменить последнее действие.
.	Повторить последнее действие.
yy	Копировать текущую строку.
yw	Копировать символы от символа в текущей позиции курсора до начала следующего слова.
p **P**	Вставить ранее скопированный текст: - правее позиции курсора; - левее позиции курсора.
J	Объединить две строки.

Для сохранения отредактированного файла и выхода из редактора необходимо ввести *ex-команду*, находясь в командном режиме. Ввод ex-команды

начинается с нажатия клавиши **«:»**. При этом в нижней части экрана должен появиться символ двоеточия. Основные ex-команды приведены в Таблице П1.2.

Таблица П1.2. Основные ex-команды **vim**

Клавиши	Команда
:w	Сохранение текущего файла.
:w *file*	Сохранение текущего файла под именем ***file***.
:wq	Сохранение текущего файла и выход из **vim**.
:q	Выход из **vim** (изменений в файл не внесено).
:q!	Выход из **vim** (без сохранения внесенных изменений).
:r *file*	Вставить в текущий файл содержимое файла ***file***.
:n **:N**	Переключение между файлами (если в **vim** редактируются несколько файлов одновременно).

Дополнительную информацию по редактору **vim** можно получить в справочном руководстве, выполнив команду:

```
$ man vi
```

nano

Консольный текстовый редактор, являющийся свободным клоном редактора **pico**, входившего в состав популярного почтового клиента **Pine**. По сравнению с **vim** имеет удобный и интуитивно понятный интерфейс, управляется сочетаниями клавиш.

Для запуска **nano** введите команду:

```
$ nano
```

Окно терминала примет вид, показанный на рис. П1.3.

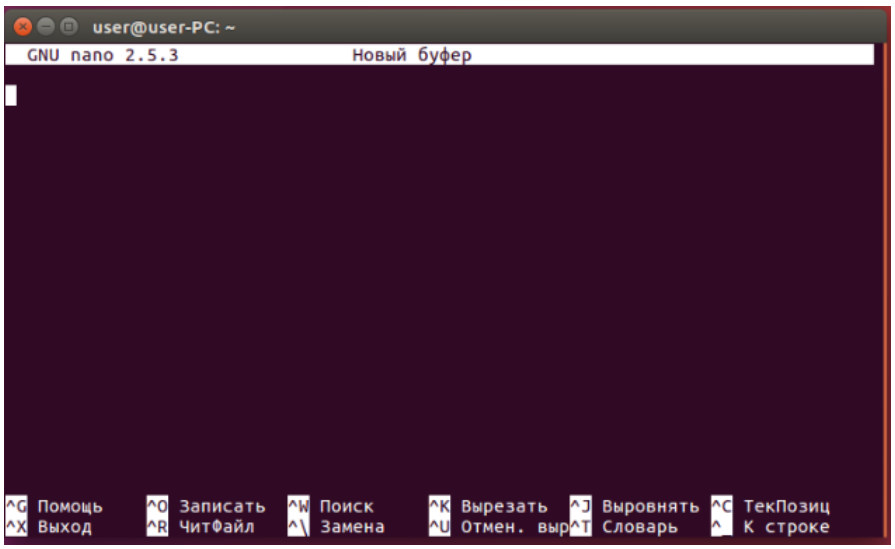


Рис. П1.3. Внешний вид терминала при запуске **nano**.

Для создания нового файла с именем `file.txt` в текущем каталоге выполним в терминале команду:

```
$ nano file.txt
```

Окно терминала при этом будет выглядеть как на рис. П1.4.

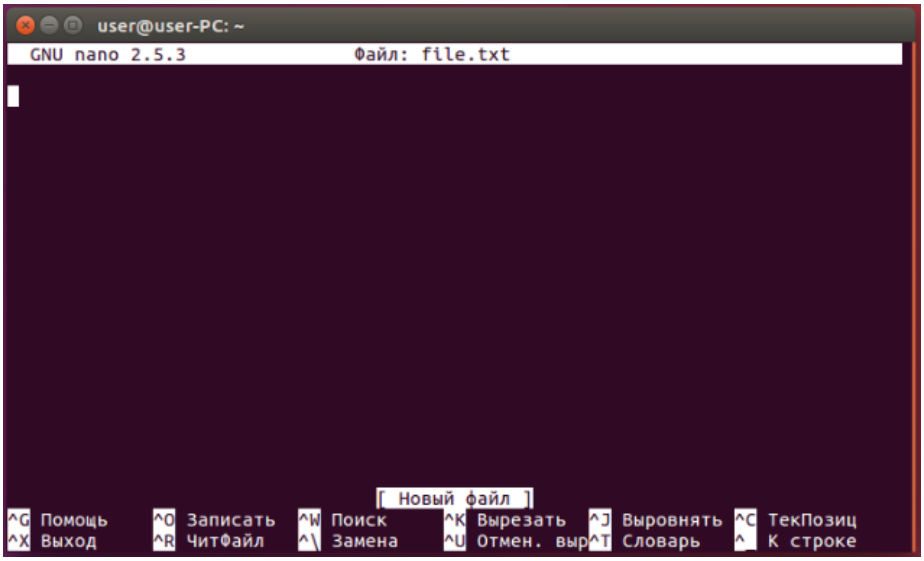


Рис. П1.4. Внешний вид терминала при создании нового файла с помощью **nano**.

Редактор разбит на 4 основные части: верхняя строка содержит версию программы, имя редактируемого файла и отображает были ли внесены изменения в текущий файл. Вторая часть – это главное окно редактирования, в котором отображен редактируемый файл. Строка состояния – 3 строка снизу – показывает разные важные сообщения. Две строки внизу показывают наиболее часто

используемые комбинации клавиш. Система обозначений комбинаций клавиш следующая. Комбинации с **«Ctrl»** обозначены символом «^» и вводятся при нажатой клавише **«Ctrl»**. Комбинации, обозначенные символом **«M»** могут быть введены при помощи нажатых клавиш **«Alt»** или **«Esc»**.

Некоторые основные комбинации для работы с редактором **`nano`** представлены в Таблице П1.3.

Таблица П1.3. Основные команды **`nano`**

Клавиши	**Команда**
«Ctrl»+G **«F1»**	Показать справку.
«Ctrl»+X **«F2»**	Закрыть текущий буфер / Выйти из **`nano`**.
«Ctrl»+O **«F3»**	Записать текущий файл на диск.
«Ctrl»+R **«F5»**	Вставить другой файл в текущий.
«Ctrl»+W **«F6»**	Искать в текущем файле текст или регулярное выражение.
«Ctrl»+K **«F9»**	Вырезать текущую строку и сохранить её в буфере обмена.
«Ctrl»+U **«F10»**	Вставить содержимое буфера обмена в текущую строку.
«Ctrl»+C **«F11»**	Показать положение курсора.
«Alt»+6	Копировать текущую строку и сохранить ее в буфере обмена.