Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

О.И. Моренкова А.Ю. Голошубов

Операционные системы.Изучаем LINUX

Практикум

Новосибирск
2022

УДК 004.451
ББК 32.973-018.2

Доцент , к.т.н. Моренкова О.И., ст. преподаватель Голошубов А.Ю. «Операционные системы. Изучаем Linux Практикум.»:Учебно-методическое пособие, СибГУТИ.- Новосибирск, 2022- 110 с.

Практикум предназначен для студентов, обучающихся по направлению 11.03.02 Инфокоммуникационные технологии и системы связи», направленность (профиль) – Системы радиосвязи, мобильной связи и радиодоступа, 11.03.02 «Инфокоммуникационные технологии и системы связи», профиль – Цифровое телерадиовещание, изучающих дисциплину «Операционные системы». В нём содержится материал, предназначенный для проведения практических или лабораторных занятий по указанному учебному курсу с целью изучения операционных систем семейства UNIX (на примере Linux).

Кафедра телекоммуникационных сетей и вычислительных средств

Рецензент: доцент Кафедры ЦТРВ и СРС ФГОУ ВО «СибГУТИ», к.т.н. Елена Викторовна Кокорева.

Утверждено редакционно-издательским советом СибГУТИ в качестве учебно-методического пособия

РЕКОМАНДАЦИИ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

Практикум по курсу «Операционные системы» посвящен изучению операционной системы LINUX.

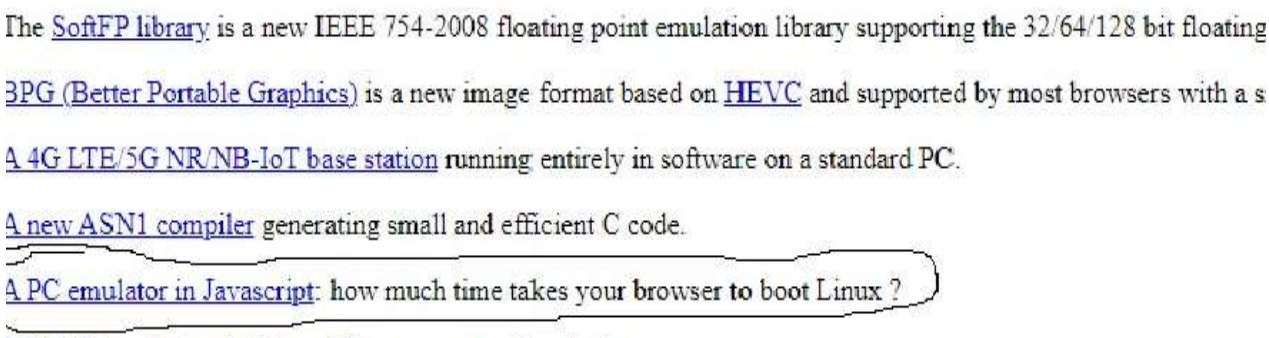
Для его выполнения необходимо установить на свой компьютер эту операционную систему. Существует много некоммерческих версий этой операционной системы (LINUX Red Hat, LINUX Mandriva, LINUX Ubuntu, LINUX Fedora и т.д.).

Можно воспользоваться on-line ОС Linux

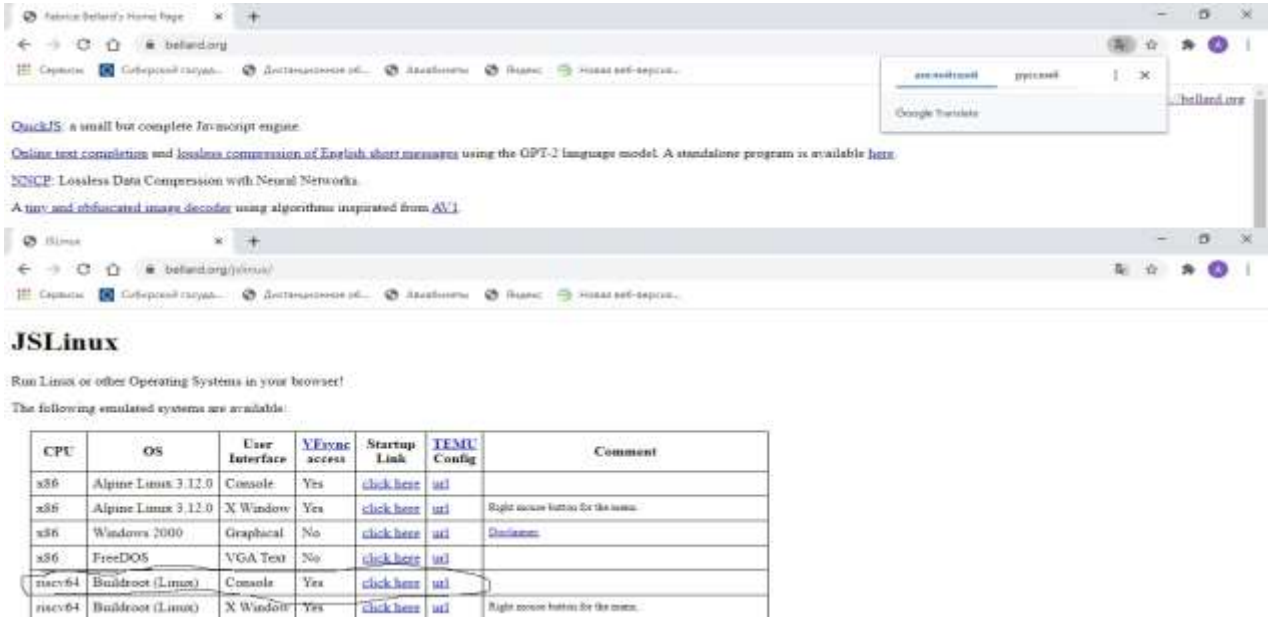
Ссылка на on-line эмулятор операционной системы LINUX

https://bellard.org

Шаг1

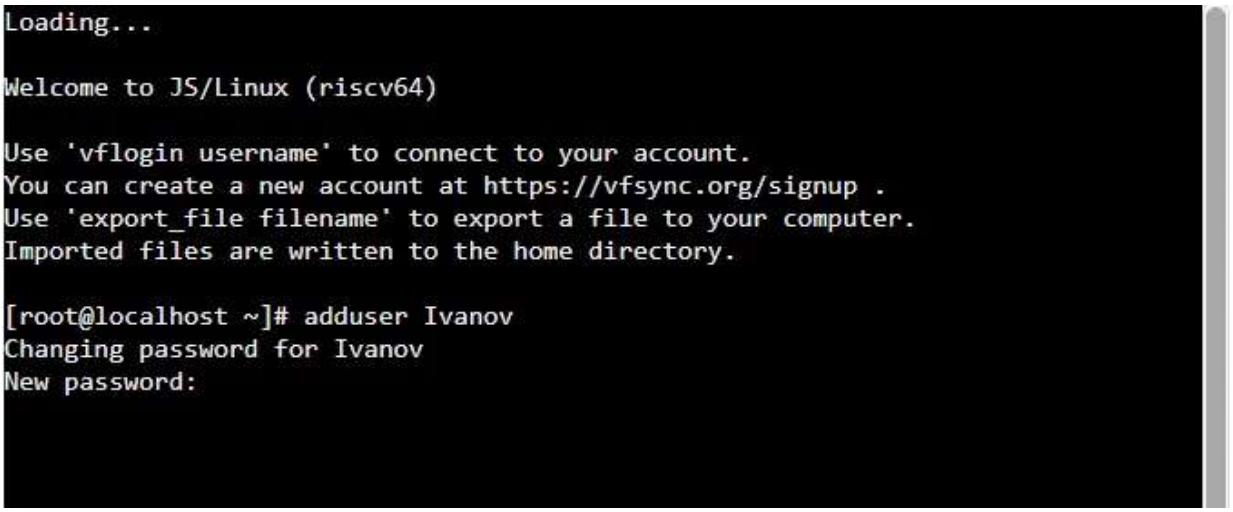


Шаг2



2

Шаг 3



Практикум состоит из семи работ. Каждая работа начинается с конспекта необходимого для ее выполнения теоретического материала. Далее прилагаются задания для лабораторной работы и требования по ее оформлению.

В конспектах теоретического материала приводится много примеров диалогов пользователя и операционной системы LINUX. Жирным шрифтом в этих диалогах выделены действия пользователя, а более светлым шрифтом – отклик системы.

Желаю успехов! Ваш преподаватель Моренкова Ольга Ильинична

Введение

Linux является клоном операционной системы UNIX. Один из наиболее интересных фактов из истории Linux'a - это то, что в его создании принимали участие одновременно люди со всех концов света - от Австралии до Финляндии - и продолжают это делать до сих пор. Сегодня Linux используется на самых разнообразных аппаратных платформах - от персональных рабочих станций до мощных серверов с тысячами пользователей. Это прежде всего потому, что Linux - это многопользовательская многозадачная система, обладающая широкими возможностями.

Linux изначально был написан Линасом Торвальдсом, а затем улучшался бесчисленным количеством людей во всем мире.

Вначале Linux разрабатывался для работы на 386 процессоре. Одним из первых проектов Линаса Торвальдса была программа, которая могла переключаться между процессами, один из которых печатал AAAA, а другой - BBBB. Впоследствии эта программа выросла в Linux.

Linux поддерживает большую часть популярного UNIX'овского программного обеспечения, включая систему X Windows. Это довольно большая программа, разработанная в Массачусетском Технологическом институте, позволяющая компьютерам создавать графические окна и используемая на многих различных UNIX'овских платформах. Linux, по большей части, совместим с System 5 и с BSD и удовлетворяет требованиям POSIX-1 (документа, пытающегося стандартизировать операционные системы). Linux также во многом согласуется с POSIX-2, другим документом IEEE по стандартизации операционных систем. Он является смешением всех трех стандартов: BSD, System 5 и POSIX.

Большинство утилит, включаемых в дистрибутивы Linux'a получены от Free Software Foundation как часть проекта GNU. Проект GNU - это попытка написать переносимую продвинутую операционную систему, которая будет выглядеть так же, как UNIX. Слово "переносимая" означает, что она будет работать на различных машинах, а не только на Intel PC, Macintosh или какой-нибудь еще.

Лабораторная работа N 1
Знакомство с операционной системой LINUX
Способы хранения информации.
Команды управления данными

Цель работы: получить базовые навыки по работе с операционной системой (ОС) Linux, ее командной оболочкой. Изучить понятия дерева каталогов, файла и типы файлов. Изучить основные команды по управлению и манипуляции данными.

1.1 Файловая система Linux.

Файлы в Linux играют ключевую роль. Данные пользователей хранятся в виде файлов. Доступ к периферийным устройствам компьютера, включая диски, CD-ROM, принтеры, терминалы, сетевые адаптеры и даже память обеспечивается с помощью файлов. Наконец, все программы, которые выполняются в системе, как системные, так и задачи пользователей, являются исполняемыми файлами. Информация на магнитных носителях (дисках, дискетах, лентах) хранится в виде файлов.
Файл - поименованная область данных па магнитном носителе. Как и во многих современных операционных системах, в LINUX файлы организованы в виде древовидной структуры (дерева), называемой *файловой системой,* которая может быть представлена как неориентированный древовидный граф, вершинам которого соответствуют файлы и каталоги. *Каталог (директория)* - элемент файловой системы, включающий в себя другие файлы и каталоги, т.е. каталог это вершина графа, имеющая больше чем одну инцидентную связь.
В Linux все доступное пользователям файловое пространство объединено в единое дерево каталогов, корневой каталог которого обозначается символом '/'. В DOS системах каждое устройство хранения данных имеет свой буквенный идентификатор (A: - Z:) и имеет свою файловую систему, корневой каталог которой обозначается '/'. Однако это не означает, что в LINUX

системе присутствует только одна файловая система. В большинстве случаев единое дерево, какое видит пользователь системы, составлено из нескольких отдельных файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим системам, могут находиться на различных устройствах.

Путь файла - это совокупность каталогов, которые надо пройти, для того чтобы получить доступ к файлу.

Пути бывают *относительные* (начало пути находится в текущем каталоге) и *абсолютные* (началом пути является корневой каталог).

Типы файлов

В UNIX системах существует 6 типов файлов, различающихся по функциональному назначению и действиям операционной системы при выполнении тех или иных операций над ними:

- Обычный файл (regular file)
- Каталог (derectory)
- Специальный файл устройства (special device file)
- FIFO или именованный канал (named pipe)
- Связь (link)
- Сокет

Обычный файл

Обычный файл представляет собой наиболее общий тип файлов, содержащий данные в некотором формате. Для операционной системы такие файлы представляют собой просто последовательность байтов. Вся интерпретация содержимого файла производится прикладной программой, обрабатывающей файл. К этим файлам относятся текстовые файлы, бинарные данные, исполняемые программы и т. п.

Каталог

Каталог - это файл, содержащий имена находящихся в нем файлов, а также

указатели на дополнительную информацию - *метаданные,* позволяющие операционной системе производить операции над этими файлами. С помощью каталогов формируется логическое дерево файловой системы. Каталоги определяют положение файла в дереве файловой системы, поскольку сам файл не содержит информации о своем местонахождении. Любая задача, имеющая право на чтение каталога, может прочесть его содержимое, но только ядро имеет право на запись в каталог.

Специальный файл устройства

Специальный файл устройства обеспечивает доступ к физическому устройству (диску, CD-ROM, floppy и т.д.). Доступ к устройствам осуществляется путем открытия, чтения и записи в специальный файл устройства. В UNIX системах различают *символьные* (character) и *блочные* (block) файлы устройств. Символьные файлы устройств используются для небуферизированного обмена данными с устройством. В противоположность этому блочные файлы позволяют производить обмен данными в виде пакетов фиксированной длины - *блоков.* Доступ к некоторым устройствам может осуществляться как через символьные, так и через блочные специальные файлы.

FIFO или именованный канал

FIFO или **именованный канал** - это файл, используемый для связи между процессами. FIFO впервые появились в System V UNIX, но большинство современных LINUX- систем поддерживают этот механизм.

Связь

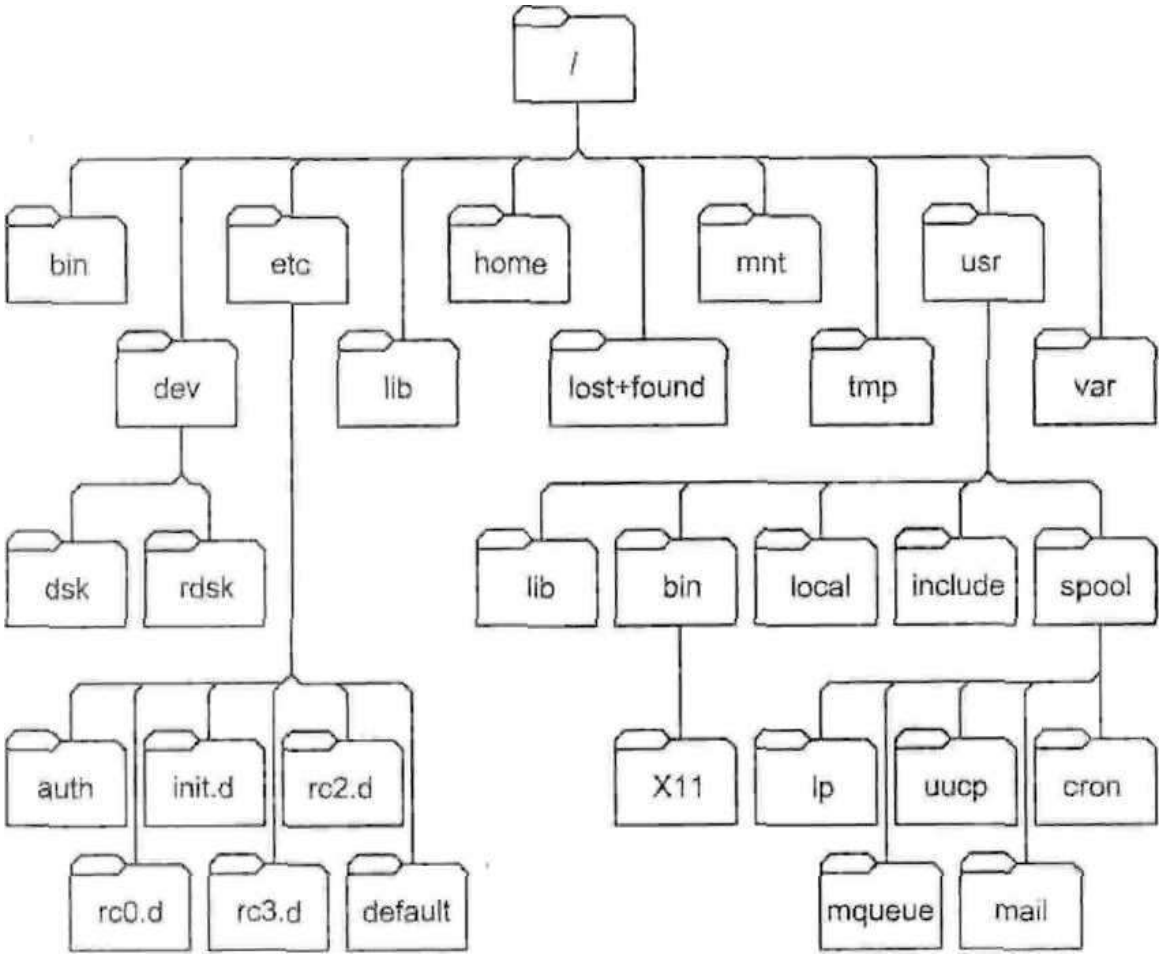
Как уже говорилось, каталог содержит имена файлов и указатели на их мета-данные. В то же время сами метаданные не содержат ни имени файла, ни указателя на это имя. Такая архитектура позволяет одному файлу иметь несколько имен в файловой системе. Имена жестко связаны с метаданными и, соответственно, с данными файла, в то время как сам файл существует

независимо от того, как его называют в файловой системе. Такая связь имени файла с его данными называется *жесткой связью* (hard link).

Сокеты

Сокеты предназначены для взаимодействия между процессами. Интерфейс сокетов часто используется для доступа к сети TCP/IP. В системах ветви BSD UNIX на базе сокетов реализована система межпроцессного взаимодействия, с помощью которой работают многие системные сервисы, например, система печати.

Файловая система LINUX.



Корневой каталог '/' является основой любой файловой системы UNIX. Все

остальные файлы и каталоги располагаются в рамках структуры, порожденной корневым каталогом, независимо от их физического местонахождения.

/bin

В каталоге /bi*n* находятся наиболее часто употребляемые команды и утилиты системы, как правило, общего пользования.

/dev

Каталог */dev* содержит специальные файлы устройств, являющиеся интерфейсом доступа к периферийным устройствам. Каталог */dev* может содержать несколько подкаталогов, группирующих специальные файлы устройств одного типа. Например, каталог */dev/dsk* содержит специальные файлы устройств для доступа к гибким и жестким дискам системы.

/etc

В этом каталоге находятся системные конфигурационные файлы и многие утилиты администрирования. Среди наиболее важных файлов - сценарии инициализации системы.

/lib

В каталоге */lib* находятся библиотечные файлы языка Си и других языков программирования. Стандартные названия библиотечных файлов имеют вид *libx.a* (или *libx.so),* где x - это один или более символов, определяющих содержимое библиотеки. Например, стандартная библиотека Си называется libc.a, библиотека системы X Window System имеет имя *libX11.a*. Часть библиотечных файлов также находится в каталоге */usr/lib*.

/lost+found

Каталог "потерянных" файлов. Ошибки целостности файловой системы, возникающие при неправильном останове LINUX или аппаратных сбоях, могут привести к появлению т. н. "безымянных" файлов. Структура и содержимое файла являются правильными, однако для него отсутствует имя в каком-либо из каталогов. Программы проверки и восстановления файловой системы помещают такие файлы в каталог */lost+found* под системными числовыми именами.

/mnt

Стандартный каталог для временного связывания (монтирования) физических файловых систем с корневой для получения дерева логической файловой системы. Обычно содержимое каталога */mnt* зависит от назначения системы и полностью определяется администратором.

/home

Общеупотребительный каталог для размещения домашних каталогов пользователей. Например, домашний каталог пользователя *student* будет называться */home/student*

/usr

В этом каталоге находятся подкаталоги различных сервисных подсистем. */usr/bin* исполняемые утилиты, */usr/local* дополнительные программы, используемые на данном компьютере, файлы заголовков */usr/include, /usr/man* электронные справочники и т.д.

/var

Этот каталог используется для хранения временных файлов различных сервисных программ (системы печати, почтового сервиса и т,п.)

/tmp

Общедоступный каталог. Используется для хранения временной информации.

1.2 Первое знакомство с LINUX

Если Вы уже установили у себя на компьютере ОС LINUX, загрузите компьютер с этой ОС.

Одной из особенностей UNIX систем, к которым относится и LINUX, является режим разделения времени, позволяющий нескольким пользователям одновременно обращаться к компьютеру. С этой целью в UNIX системах предусмотрена поддержка индивидуальной рабочей среды. Система идентифицирует каждого пользователя, следит за тем, когда он зарегистрировался в системе, сколько времени проработал в ней, контролирует доступ к файлам и сетевым ресурсам, тип используемого терминала и т.д.

После того, как закончилась процедура загрузки операционной системы LINUX, вы увидите что-нибудь вроде:

```
Blac Cat Linux release 6.2
Kernel 2.2.19-4bc on an i686
Welcome to wpl.

 wpl login:
```

Однако, возможно, система, используемая вами, представится по-другому (например, если после загрузки система автоматически переключается в графический режим регистрации). Тем не менее, она тоже попросит вас пройти процедуру идентификации, и будет функционировать во многом похоже. Переключение между графическим и текстовым режимами обычно производится комбинацией клавиш <CTRL+ALT+F?>, где F? означает одну из функциональных клавиш F1 - F6, либо «/система/ терминал».

Итак - приглашение для login. Для примеров мы будем использовать фиктивного пользователя *student.* Везде, где вы увидите *student,* подразумевается ваше пользовательское имя.

Регистрационное, или *пользовательское имя* применяется для идентификации пользователя (абонента) системы. Оно назначается системным администратором при создании регистрационной записи. В качестве такого имени могут выступать не только ваши инициалы или полное имя, но и любая произвольная последовательность символов (например, регистрационное имя для студента может иметь вид pl0s40).

Внимание: В именах, используемых в UNIX системах, учитывается регистр символов. Поэтому под именами *Student* и *student* могут работать два совершенно разных пользователя. Это справедливо и для паролей.

После ввода *student,* вы увидите следующее:

```
Black Cat Linux release 6.2
Kernel 2.2.19-4bc on an i686
Welcome to wpl.
```

wpl login: **student**

password:

Это Linux спрашивает ваш пароль. Когда вы вводите ваш пароль, вы не будете видеть, что вы набираете. Набирайте осторожно, пароль можно стереть (используя комбинацию клавиш <CTRL+W>), но вы не будете видеть, что вы редактируете. Если вы сделали опечатку, вам будет представлена новая возможность для login'a.

Black Cat Linux release 6.2
Kernel 2.2.19-4bc on an i686
Welcome to wpl.

wpl login: **student**
password: Login incorrect
wpl login:

Если вы ввели свои имя и пароль правильно, система будет готова к работе. **После регистрации в системе, LINUX делает текущим ваш домашний каталог и запускает программу оболочки. Если вы используете графический интерфейс (GUI), то после регистрации на экране будет отображаться рабочий стол. Если вам необходимо использовать командную оболочку, начните сеанс работы в режиме эмуляции терминала («система/ терминал».)**

После перехода в окно терминала появится короткое сообщение, называемое сообщением дня. Оно может быть любого содержания или вообще не иметь его, это решает системный администратор. После этого на экран будет выдано приглашение (возможно такое)

[student@wpl student]$

С вами начинает общение командная оболочка ОС LINUX.

1.3 Командная оболочка

Вывод строки приглашения и мерцание курсора на экране означает, что система готова к приему команд.

Командная оболочка **- это программа, предназначенная для обработки, преобразования и выполнения команд, введенных пользователем.** Под обработкой и преобразованием понимается набор действий командной оболочки по интерпретации и исполнению команд пользователя.

В настоящее время в UNIX системах применяются различные оболочки. Наиболее популярными являются:

- **Bourne shell (sh)**, названная в честь своего создателя Стивена Борна (Steven Bourne) из AT&T Bell Labs
- **Bourne Again Shell (bash)**, расширенная версия предыдущей оболочки-**C shell (csh**), разработанная Билом Джоем (Bill Joy), первоначально была создана BSD UNIX, сейчас входит в состав System V.
- **Korn shell (ksh)**, созданная Дэвидом Корном (David Korn) на базе оригинальной bash, но также реализующая некоторые возможности оболочки C.

В LINUX есть несколько вариаций этих оболочек. Две наиболее часто используемые, это новый Shell Борна (Bourne Again Shell) или "sh" или bash. Bash - это развитие прежнего shell с добавлением многих полезных возможностей, частично содержащихся в C shell. Поскольку bash можно рассматривать как надмножество синтаксиса прежнего shell, любая программа, написанная на добром старом shell Борна должна работать и в bash. Для тех, кто предпочитает использовать синтаксис C shell, LINUX поддерживает bash, который является расширенной версией C shell.
Вы можете работать в системе, не зная о том, какая оболочка используется Вами. Однако опытный пользователь может догадаться об этом по стандартному приглашению командной строки. По умолчанию в оболочках используется перечисленные ниже символы, приглашающие ввести очередную команду, однако при желании вы можете легко их изменить.

- Bourne использует символ "$"
- C shell - знак процента ("%")
- Korn shell - тот же символ, что и Bourne shell ("$")

Существуют и другие способы выяснить тип используемой оболочки. **Файл *<u>/etc/passwd</u>* содержит информацию обо всех пользователях, зарегистрированных на данном локальном компьютере.**

Каждая строка этого файла описывает одного из пользователей системы. Формат строки этого файла следующий (поля разделяются символом **:**):

student::500:500:Иванов Петр Афонасьевич:/home/student:/bin/bash*

1. Системное имя пользователя *(student).*
2. Зашифрованный пароль (*).
3. Идентификационный номер пользователя UID *(500).*
4. Идентификационный номер основной группы пользователя GID *(500).*
5. Имя пользователя *(Иванов Петр Афонасьевич).*
6. Домашний каталог *(/home/student).*
7. Командная оболочка ***(/bin/bash).***

Используя информацию из этого файла можно легко определить, какую оболочку вы используете (поле 7).

И так, вывод строки- приглашения и мерцание курсора на экране означает, что система готова к приему команд.

Командой называется символьная строка, вводимая пользователем для управления операционной системой и завершаемая символом перевода каретки (клавиша <ENTER>). Каждая команда состоит как минимум из одного поля - *имени команды,* представляющее собой имя файла, который требуется

выполнить. Если путь к файлу явно не указан, то командная оболочка попытается найти этот файл в одном из известных ей каталогов и выполнить его. Файлом может быть либо программа, либо текстовый файл, содержащий команды оболочки. В первом случае командная оболочка создаст процесс и загрузит туда названный файл. Во втором случае командная оболочка интерпретирует и выполнит команды из этого файла.

1.4 Команды управления данными

Приступим к изучению ряда полезных, а, иногда, просто необходимых команд управления операционной системой LINUX. Для использования этих команд следует перейти из графического режима ОС LINUX в режим эмуляции терминала **«переход»/система/ терминал»**.

Определение текущего каталога

Имя текущего каталога, т.е. каталога, с которым вы работаете в данный момент, является, пожалуй, самой важной информацией о вашей рабочей среде. Для определения имени текущего каталога можно использовать команду **pwd.**
В системе LINUX файлы хранятся в разных каталогах. В любой момент только один каталог является текущим, и команды оболочки по умолчанию применяются к файлам или подкаталогам этого каталога. После регистрации в системе, текущим будет ваш рабочий каталог. Вы вправе выполнять любые операции над содержащимися в нем файлами и подкаталогами.

Для перемещения из одного каталога в другой служит команда **cd,** описанная далее. Перед выполнением какой-либо команды необходимо убедиться, что вы находитесь в нужном каталоге.

Внимание: Поскольку в системе LINUX имена файлов задаются с учетом регистра, в вашей системе вполне можете оказаться два каталога, имена которых

различаются лишь регистром букв, например, PROJECT и project. В результате, по ошибке, легко попасть в каталог project, в то время как на самом деле нужен каталог PROJECT. В связи с этим, перед применением любой команды рекомендуется использовать команду **pwd,** которая позволит удостовериться, что вы находитесь в нужном каталоге.

Совет: Вообще, неплохо бы выработать для себя определенный порядок именования каталогов. Например, некоторые пользователи применяют в именах каталогов прописные буквы, а в именах файлов - строчные. Желательно не использовать в именах файлов русские буквы (из-за проблем с русскими кодировками в разных системах)

Сразу после регистрации в системе вы оказываетесь в рабочем каталоге. Предположим, он называется */home/student*. Если вы выполните команду **pwd** непосредственно после того, как войдете в систему, на экране отобразится имя */home/student*.

[student@wpl student]$**pwd**<Enter>
/home/student
[student@wpl student]$

Изменение текущего каталога

С помощью команды **cd** вы можете сделать текущим любой указанный вами каталог. Например

[student@wpl student]$**pwd**<Enter>
/home/student
[student@wpl student]$**cd /home/abcd**<Enter>
[student@wpl abcd]$ **pwd**<Enter>
/home/abcd
[student@wpl abcd]$

Внимание: Если вы хорошо знакомы с системой DOS, то заметите, что команда **cd** работает в LINUX практически так же, за исключением двух

особенностей. Если в системе DOS вы используете команду **cd** без параметров, на экране отобразится имя текущего каталога (как при выполнении команды LINUX **pwd**). Но если при вызове команды **cd** в системе LINUX вы не укажете имени каталога, то текущим станет ваш домашний каталог. Второе отличие в том, что в DOS для разделения каталогов используется обратная косая черта (\), а в LINUX - обычная косая черта ("/").

Совет: Вы должны твердо запомнить, что для разделения каталогов в LINUX используется обычная косая черта '/', а не обратная '\', к которой вы привыкли в DOS, Обратная косая черта в LINUX выполняет особую функцию. Более подробно этот вопрос будет рассмотрен далее.

В качестве параметра команды **cd** может быть задан либо абсолютный путь (как в предыдущем примере), либо относительный (как в примере, приведенном ниже).

[student@wpl student]$**pwd**<Enter>
/home/student
[student@wpl student]$**cd ../abcd**<Enter>
[student@wpl abcd]$**pwd**<Enter>
/home/abcd
[student@wpl abcd]$

Две точки (псевдоним родительского каталога) в первой команде **cd** перемешают вас на один уровень вверх, после чего вы смещаетесь вниз, в дочерний каталог с именем *abcd.* Вы можете переместиться сразу на несколько уровней вверх по дереву каталогов, задавая имя '..' соответствующее количество раз (например, если вы введете

cd ../../, то перейдете сразу на два уровня вверх).

Внимание: Если имя ".." означает родительский каталог, то символ "." указывает на текущий каталог.

Вывод информации о содержимом каталога

Команда **ls** выводит информацию о содержимом каталога. В качестве параметра команды **ls** может быть задано либо имя файла, либо имя каталога. Кроме того, эта команда обрабатывает ряд ключей. Ключи определяют внешний вид списка файлов и информацию, отображаемую для каждого из них. Если в качестве параметра будет указано имя каталога, то ключи команды **ls** будут влиять на вывод информации для всех файлов, хранящихся в этом каталоге (сказанное верно только в том случае, если опция *-d* не задана).

Часто используемые ключи команды **ls** перечислены в табл. 1.

Таблица 1- Ключи команды **ls**

ключ	Описание
-a	Выводит информацию обо всех файлах. По умолчанию команда ls не отображает файлов, имена которых начинаются с точки ("."), а ключ -a делает это возможным
-d	Выводит информацию только для каталога, но не для его содержимого.
-F	Отображает после имен каталогов косую черту ("/"), после имен исполняемых файлов - звездочку ("*") и после символьных ссылок - знак "@"
-i	Выводит в первом столбце списка индексный дескриптор файла. Если один из файлов является ссылкой на другой, оба будут иметь один и тот же индексный дескриптор
-l (эль)	Представляет список файлов в виде одного столбца и выводит детальную информацию о каждом файле: размер в байтах, имя его владельца, права доступа к нему и т.д. Если данный ключ не указан, выводятся только имена файлов
-n	Отображает вместо имен пользователей и групп их идентификационные номера
-r	Изменяет порядок сортировки на обратный

-S	Отображает размеры файлов (учитывая косвенные блоки) в 512-байтовых блоках (System V).
-t	Располагает файлы по дате их изменения. Если содержимое файла изменялось недавно, он будет отображаться в начале списка
-u	Сортирует файлы по времени последнего обращения к ним
-R	Показывает содержимое каталога и всех его подкаталогов

Если при вызове команды **ls** не указано ни имя файла, ни имя каталога, то выводится содержимое текущего каталога. По умолчанию имена отображаются в алфавитном порядке, например:

[student@wpl student]$**ls**<Enter>

Desktop Hail labl.c lab2.txt nsmail
[student@wpl student]$

Ключи команды ***ls*** могут быть указаны как по отдельности, так и в виде одной последовательности символов. Это означает, что выражения

ls -l –F и
ls –lF дадут одинаковый результат.

Копирование файлов и каталогов

Утилита **cp** позволяет делать копии любых файлов - как текстовых, так и исполняемых файлов, содержащих программы. Для успешного выполнения программы **cp** необходимо иметь право чтения файла, а также право записи в каталог, в который вы собираетесь скопировать файл. Поэтому, если вам не удается выполнить программу **cp,** проверьте права доступа к исходному файлу и к каталогу назначения, используя

команду **ls –l.**

Внимание.: Если вы записываете файл поверх существующего, вам также необходимо иметь право на запись в файл.

При вызове команды **cp** указываются исходный файл и файл назначения.

[student@wpl student]$**cp** исходный-файл файл-назначения

где исходный файл - имя копируемого файла, а файл назначения - имя копии файла, которая должна быть получена с помощью утилиты **cp**. Если в качестве файла-назначения указано имя каталога, команда **cp** скопирует исходный файл в этот каталог.

Перемещение и переименование файлов

В системе LINUX перемещение и переименование файлов осуществляется посредством одной и той же команды - **mv.** Основные правила ее вызова такие же, как и для команды копирования **cp.** Вы можете перемещать любое количество файлов в каталог, но имя каталога должно быть указано последним в команде, и вы должны иметь право записи в этот каталог.
Однако в отличие от утилиты **cp**, команда **mv** позволяет перемещать и переименовывать каталоги. При перемещении или переименовании файла изменяется лишь запись в каталоге (если, конечно, файл не перемещается на другой физический носитель или другой раздел диска; в этом случае переписывается не только содержимое каталога, но и содержимое файла).

Создание каталогов

В LINUX каталоги создаются с помощью команды **mkdir,** которая вызывается следующим образом:

[student@wpl student]$**mkdir** *имя_каталога*

, где *имя каталога* - имя, которое вы присваиваете новому каталогу.
В процессе создания каталога система автоматически создает в нем две

записи, описывающие каталоги с именами "." и "..". Такие записи содержатся во всех каталогах, и пользователи не вправе их удалять. Каталог, который содержит только пункты "." и "..", считается пустым.

Удаление файлов и каталогов

Команда **rm** позволяет удалить файл. Для ее выполнения надо иметь право записи в каталог, в котором находится удаляемый файл. Прав на чтение файла или запись в него не требуется. Объясняется это тем, что команда лишь удаляет соответствующую запись из каталога, поэтому права на изменение каталога достаточно для удаления содержащихся в нем файлов. Для удаления каталога прежде следует удалить все его содержимое (включая все подкаталоги со всем их содержимым), и только после этого использовать команда **rmdir:**

[student@wpl student]$**rmdir имя_каталога**

Для удаления каталога и всех содержащихся в нем подкаталогов и файлов можно воспользоваться специальным ключом **-r** команды **rm.**

[student@wpl student]$**rm -r** имя_каталога

Внимание: При использовании команды **rm** будьте внимательны, поскольку удаленные файлы действительно исчезают. Единственный способ вернуть информацию - обратиться к системному администратору и восстановить файл, воспользовавшись последней резервной копией (версия файла, хранящаяся в ней, не обязательно окажется самой "свежей"). Поэтому, если у вас остаются сомнения, воздержитесь от удаления файла.

Создание связи

С помощью команды **ln** мы можем создать еще одно имя *(second)* файла, на который указывает имя *first:*

[student@wpl student]$**ls**<ENTER>
first
[student@wpl student]$**ln first second**<ENTER>
[student@wpl student]$**ls**<ENTER>

first second

Жесткие связи абсолютно равноправны. В списках файлов и каталогов, которые можно получить с помощью команды ls, файлы first и second будут отличаться только именем. Все остальные атрибуты файла будут абсолютно одинаковыми. С точки зрения пользователя это два разных файла. Изменения, внесенные в любой из этих файлов, затронут и другой, поскольку оба они ссылаются на одни и те же данные файла. Вы можете переместить один из файлов в другой каталог, все равно эти имена будут связаны жесткой связью с данными файла. Легко проверить, что удаление одного из файлов *(first* или *second)* не приведет к удалению самого файла, т.е. его метаданных и данных (если это не специальный файл устройства). Данное утверждение верно лишь отчасти. Действительно, файлу 'безразлично', какие имена он имеет в каталогах, но 'небезразлично' число этих имен. Если ни одно из имен файловой системы не ссылается на файл, он должен быть удален (т. е. физически удалены его данные на диске).

Вывод содержимого файла на экран

Вывести на экран содержимое файла можно командой **cat.** Например, набор команды

[student@wpl student]$**cat report.txt** <Enter>

приведет к выводу на экран содержимого файла report.txt.

Определение типа командной оболочки

Если, используя файл */etc/passwd* не удалось узнать какую оболочку вы используете, то это можно сделать, введя команду **ps**, которая выводит информацию обо все процессах (программах), запущенных на вашем компьютере.

```
[student@wpl student]$ps<Enter>
661 pts/0       00:00:00 bash
907 pts/0       00:00:00    ps
 [student@wpl student]$
```

Не вдаваясь пока в подробности работы *команды* **ps,** можно сказать, что первая строка говорит об использовании оболочки **bash**, вторая - о выполнении самой команды **ps**.

Команда `touch`

В Linux команда touch предназначена для изменения метки времени. Утилита используется также для создания пустых файлов, хотя это не основная её функция. Команда touch может изменить модификацию и время доступа для любого заданного файла. Утилита создает файл только в том случае, если он ещё не существует.

Создание файла

Создание пустого файла — самый простой способ использования команды `touch` без параметров:

```
student@ mobile:~/Directory:$ touch example
student@mobile:~/Directory:$ ls -l
итого 0
-rw-r--r-- 1 oleg users 0 июл 26 11:40 example
student@mobile:~/Directory:$
```

Создание нескольких файлов

Для создания нескольких файлов командой `touch` нужно лишь перечислить их имена, разделив их пробелами:

```
student@mobile:~/Directory:$ touch example_1 example_2 example_3
student@mobile:~/Directory:$ ls -l
итого 0
-rw-r--r-- 1 oleg users 0 июл 26 11:42 example_1
-rw-r--r-- 1 oleg users 0 июл 26 11:42 example_2
-rw-r--r-- 1 oleg users 0 июл 26 11:42 example_3
```

```
student@mobile:~/Directory:$
```

Создание большого пакета файлов

Для создания большого количества файлов командой `touch` необходимо к имени файла добавить первый и последний элемент, заключив их в фигурные скобки и отделив друг от друга двумя точками (..):

```
student@mobile:~/Directory:$ touch example_{1..15}
student@mobile:~/Directory:$ ls -l
итого 0
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_1
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_10
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_11
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_12
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_13
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_14
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_15
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_2
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_3
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_4
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_5
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_6
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_7
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_8
-rw-r--r-- 1 oleg users 0 июл 26 11:48 example_9
```

Более подробную информацию о любой из представленных команд можно получить, используя электронный справочник **man**. Например, информация о команде **ls** может быть получена командой
[student@wpl student]$ **man ls** <*Enter*>

Маска файлов.

Нередко ту или иную команду требуется применить не к одному файлу, а сразу к целой группе файлов. Для этой цели при вызове команды в имена файлов включаются специальные символы, которые называются символами расширения. С помощью этих спецсимволов формируется так называемая маска этой группы имен файлов. Оболочка преобразует имя,

содержащее символы расширения, в список имен и передает этот список команде, указанной в командной строке.

Символ расширения "*"

Знак "звездочка" применяется чаще всего. Например, последовательность *a** определяет все файлы, имена которых начинаются с буквы "a". Символ '*' можно использовать в любом месте имени. Например, выражение **xx*.dat* определяет все файлы, оканчивающееся символами .d*at,* в имени которых присутствуют буквы "*xx*". Это могут быть имена *abxx.dat, lxx33.dat, xxyyzz.dat* или даже *xx.dat.*

Символ расширения "?".

Знаком вопроса представляется вхождение одного любого символа в имя файла Таким образом, последовательность ??? определяет все файлы, имена которых состоят только из трех символов. Сгенерировать список имен, оканчивающихся тремя символами, отделенными от остальной части имени точкой, позволяет выражение .???. Например, если вы осуществляете поиск файлов, оканчивающихся точкой и тремя символами, к числу которых относятся *файлы . tif . jpg .exe* вам необходимо набрать следующую команду (поиск будет осуществляться в текущем каталоге).

[student@wpl student]$**ls *. ???**<Enter>

Выражение { }.

В некоторых случаях символов расширения недостаточно и требуется более избирательный подход к генерации списка имен файлов. Допустим, вам необходимо выбрать файлы jobl, job2, job4, исключив при этом job3 или jobx. Символ ? в этом случае не поможет, потому что он соответствует единичному вхождению **любого** символа. В этой ситуации следует ввести выражение *job{124}.* В результате поиска по заданной таким образом маске будут выявлены все файлы, в названии которых после слова job будет фигурировать одна из указанных в фигурных скобках цифра.

Используя фигурные скобки, можно отобрать необходимые комбинации из

числа тех, что представляют для вас интерес, например{124}. Ничто не мешает вам задать в качестве критерия отбора диапазон символов, например {A..Z}, которые соответствуют всем прописным буквам от A до Z включительно. С помощью фигурных скобок и знака '..' вы можете указать в качестве критерия отбора сразу несколько диапазонов символов – {1..3, 5..8, a..e, A..E}.

Интересно отметить, что как только символ '..' оказывается за пределами фигурных скобок, он перестает выполнять роль метасимвола, тогда как звездочка и вопросительный знак выступают в качестве символа расширения, только находясь за пределами квадратных скобок. Таким образом, выражение ..[*?]abc будет определять всего лишь два вида имени: *abc и ?abc.

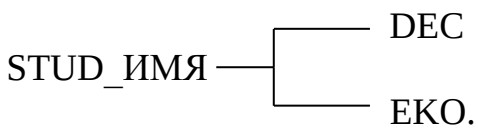
Задание для лабораторной работы №1

Общее задание для всех:

1. Если вы еще не установили операционную систему LINUX, установите или воспользуйтесь ссылкой на on-line эмулятор оболочки Linux..
2. Войти в систему LINUX . Если система требует, пройдите процедуру идентификации.
3. Выберите на панели монитора режим «терминал».

Учимся создавать новый каталог.

4. Убедитесь, что Вы находитесь в своем домашнем каталоге.
5. Создать в своем домашнем каталоге подкаталог STUD_ИМЯ следующей структуры:



6. Переименовать подкаталог EKO в MPM.

Учимся создавать новый файл.

7. Сделать текущим каталог MPM.
8. nano (вызов встроенного редактора для создания нового файла).
9. В открывшемся окне наберите любой текст, но не менее чем из пяти строк. Например: абзац из учебника, письмо другу, стихи и т.п.
10. Воспользуйтесь подсказкой меню внизу экрана для записи файла на диск. Файл назовите своим собственным именем.
11. Воспользуйтесь подсказкой меню внизу экрана для выхода из редактора.
12. Просмотрите содержимое созданного вами файла. Для этого воспользуйтесь командой вывода содержимого файла (см. теорию)

Учимся редактировать файл.

13. Отредактировать созданный вами текст. Для этого необходимо:

Вызвать встроенный редактор с указанием имени вашего файла, например:

nano Anna

внесите изменения в созданный вами ранее текст:

a) добавьте в конец текста еще одну строку;
b) вставьте новую строку между второй и третьей строками;
c) в первой строке сотрите любое слово;
d) при обнаружении допущенных ранее ошибок, исправьте их.

-воспользуйтесь подсказкой меню внизу экрана для сохранения исправленного текста под новым именем;

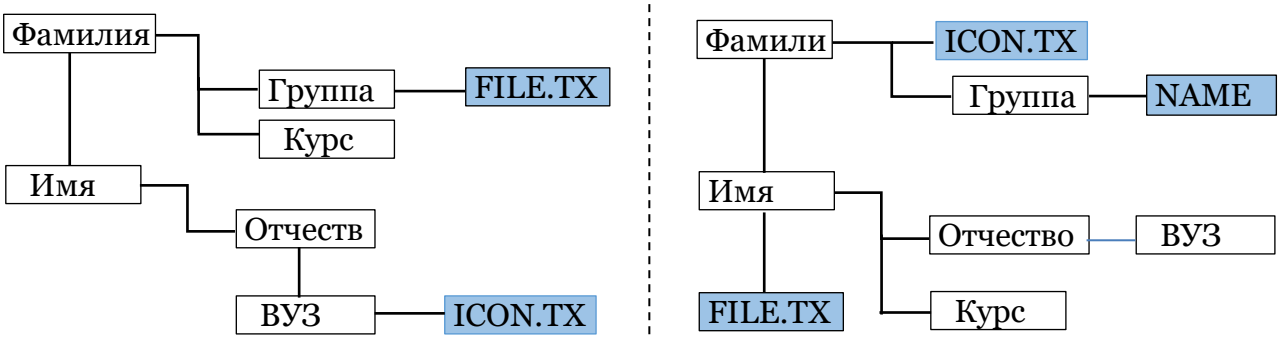
Воспользуйтесь подсказкой меню для выхода из режима редактирования.

14. Просмотрите содержимое созданного вами файла. Для этого воспользуйтесь командой вывода содержимого файла (см. теорию)
15. Создать копию последнего файла в каталог DEC.

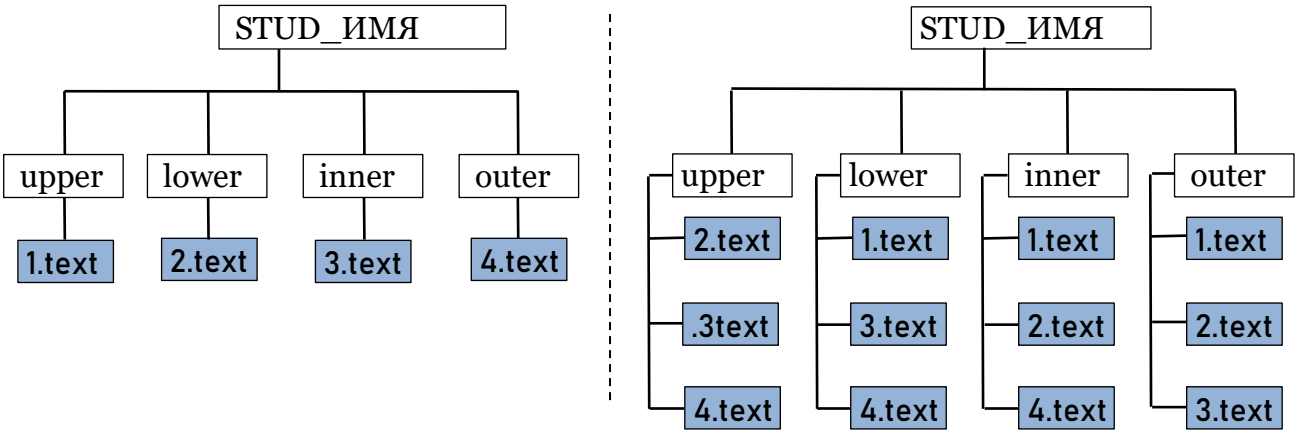
Индивидуальные задания

ЗАМЕЧАНИЕ: бесцветные прямоугольники- каталоги;
Цветные – обычные файлы;

Вариант 1.
Создать в домашнем каталоге структуру каталогов (а). Из нее получить (б).

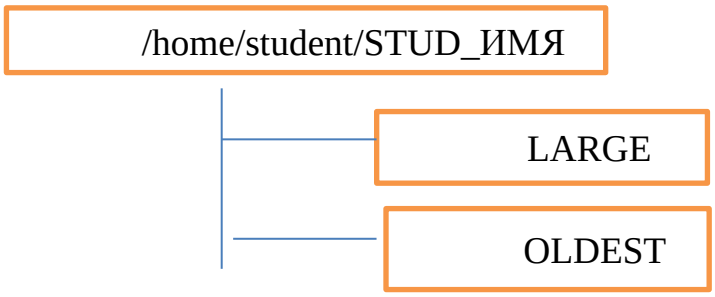


Вариант 2.
Находясь в корневом каталоге и воспользовавшись командой группировки, создать в домашней директории подкаталоги upper, lower, inner, outer. В каждом из них создать по одному текстовому файлу с окончанием .text (а). Создать копию каждого из файлов в других каталогах. Удалить оригинальные файлы (б). Используя символы подстановки, отобразить свойства только этих файлов в расширенном формате.



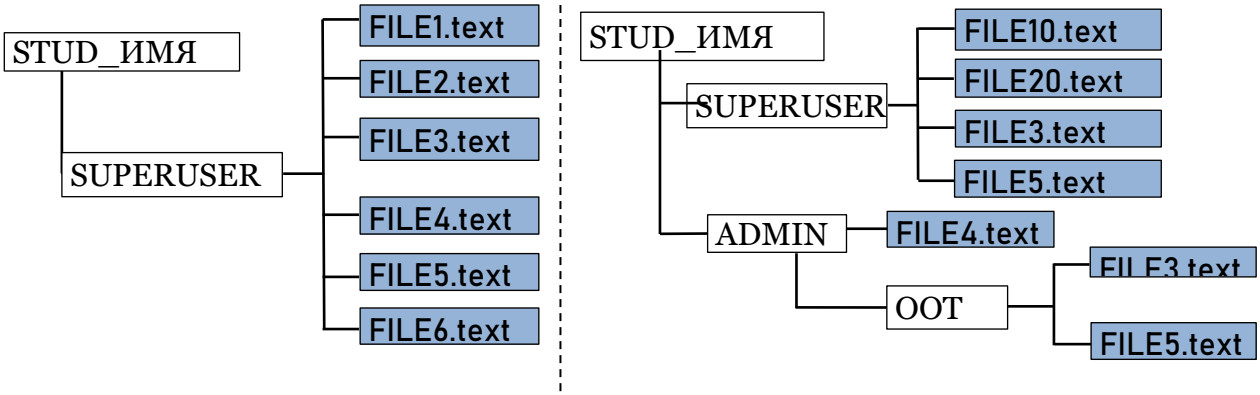
Вариант 3.

В домашнем каталоге создать структуру (на рисунке). Максимальный по размеру файл (включая скрытые) из каталога **/tmp** записать в LARGE. Первые три файла из каталога **/var/log**, которые были созданы раньше остальных, скопировать в OLDEST. Выдать оглавление обоих каталогов в рекурсивном виде.



Вариант 4.

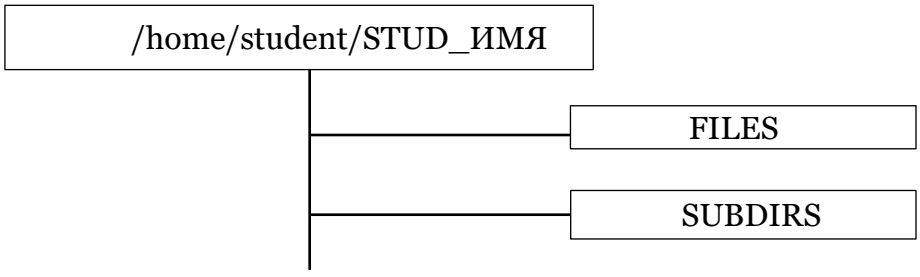
В домашней директории создать подкаталог SUPERUSER. С помощью команды группировки создать в нем пустые файлы FILE1.text, FILE2.text, FILE3.text, FILE4.text, FILE5.text, FILE6.text (а). Заполнить их произвольным текстом. Выполнив операции копирования, переименования, удаления,



получить структуру (б).

Вариант 5.

Определить среди подкаталогов домашнего каталога тот, который изменялся раньше остальных. Если он оказался пуст, то определить каталог, доступ к которому был осуществлен раньше других. Создать структуру (на рисунке).

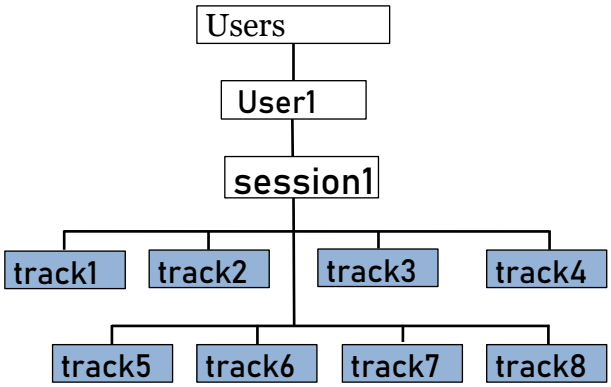


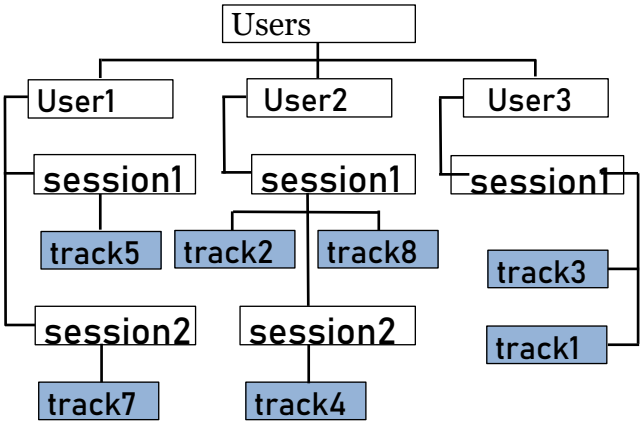
Файлы из него скопировать в FILES, а подкаталоги в SUBDIRS **.**

Вариант 6.

Создать структуру каталогов (а). Получить (б).

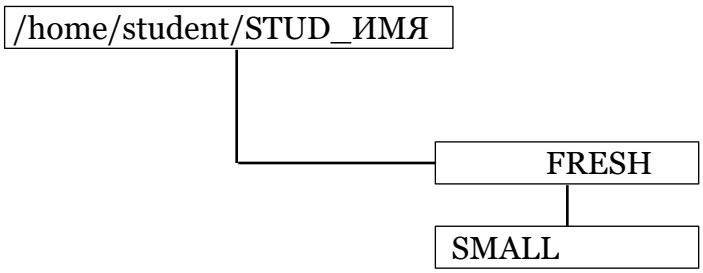
Выдать оглавление Users в рекурсивном виде.





Вариант 7.

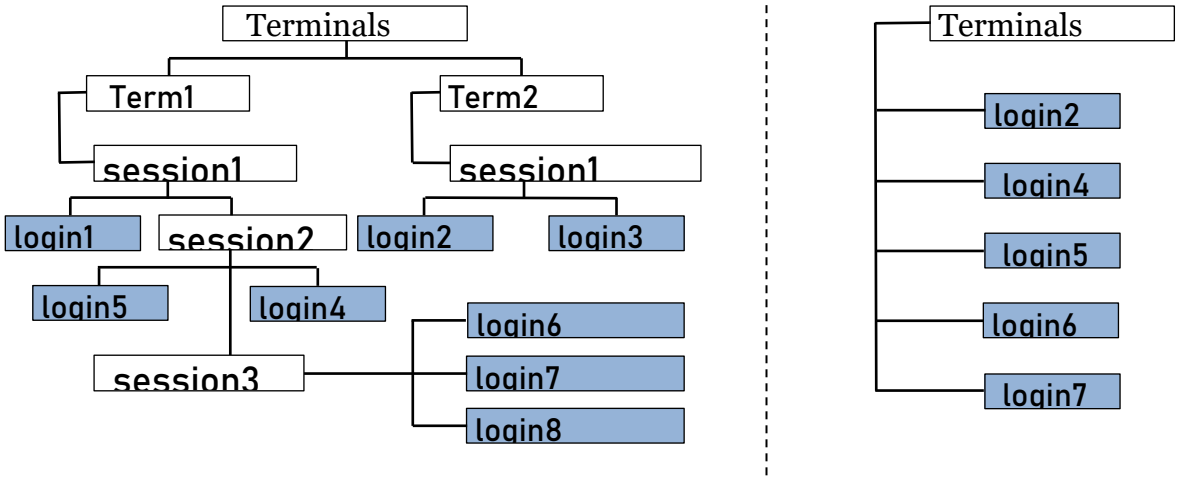
В домашнем каталоге создать структуру (на рисунке). Первые три файла (символьные ссылки не трогать) из каталога **/bin**, которые занимают меньше всего места, скопировать в SMALL. Самый свежий (по дате последнего изменения) файл из каталога **/usr/lib/** записать во FRESH. Выдать оглавление FRESH в рекурсивном виде.



Вариант 8.

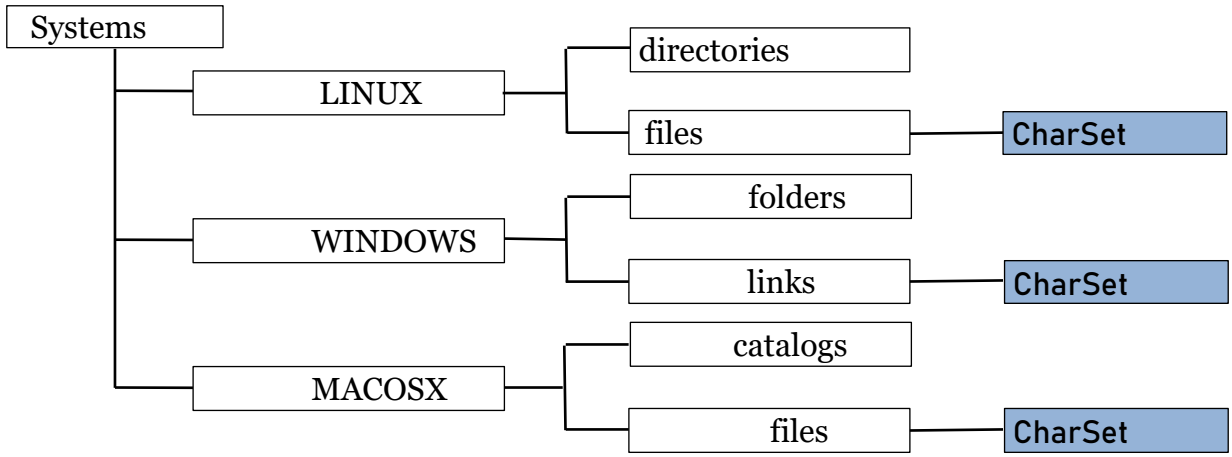
Создать структуру каталогов (а). Получить (б).

Выдать оглавление Terminals в порядке, отсортированном по дате изменения.



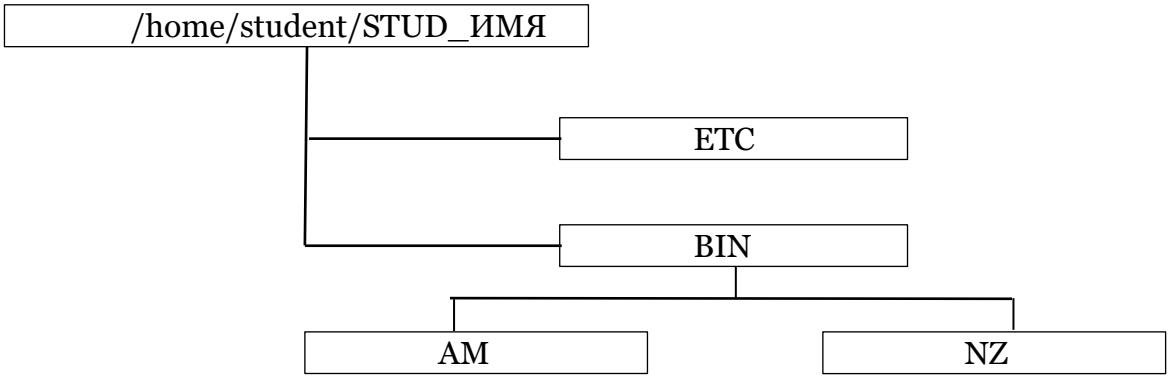
Вариант 9.

В домашнем каталоге создать подкаталог Systems со структурой на рисунке.



Переименовать файлы CharSet в UTF-8, ANSI-1251, MacCyrillic соответственно. Сделать их копию в каталоге Systems.

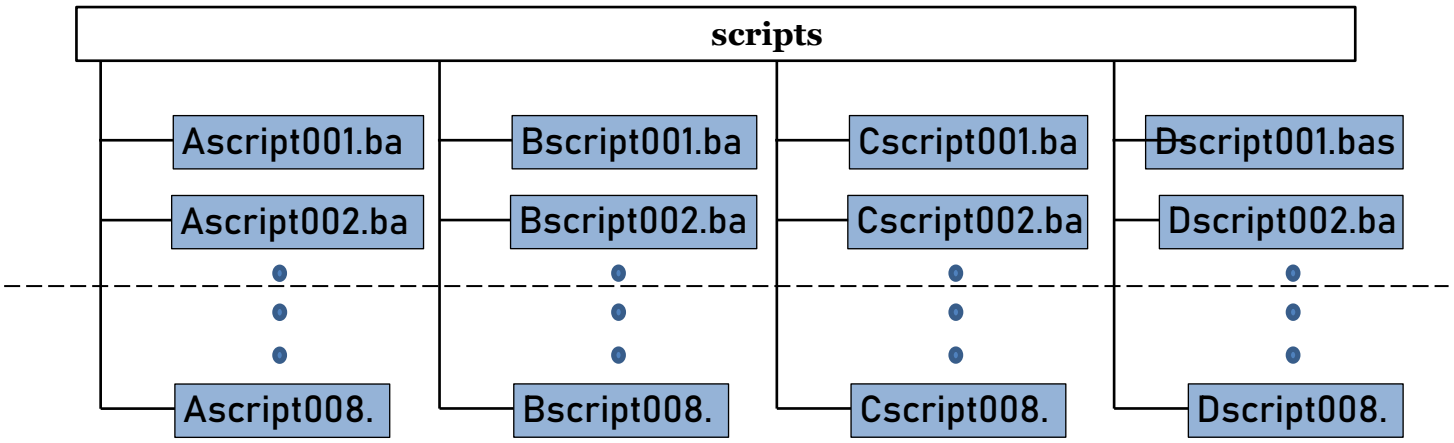
Вариант 10.



Применив маску, в каталоге **/etc** найти файлы, оканчивающиеся на **“conf”.** Записать их в подкаталог **ETC**. В каталоге **/bin** определить файлы, состоящие из семи символов. Те из них, что начинаются с **a, b, … m,** записать в **AM,** остальные в **NZ.**

Вариант 11.

Все операции с каталогами и файлами выполнять, находясь в **/tmp.** В директории **/tmp** создать подкаталоги A, B, C, D, OTHERS. В домашнем каталоге создать структуру (на рисунке) с применением команды группировки. Каждый пятый файл в группе заполнить произвольным текстом. Эти 4 файла записать в OTHERS, начинающиеся с A — в A и т.д., оригинальные файлы удалить. Использовать команду группировки и маску файлов.



Требование к отчету о лабораторной работе

Отчет по лабораторной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель лабораторной работы
3. Полный протокол выполнения индивидуального задания лабораторной работы с пояснением каждой команды. Вид командной строки и результаты работы каждой команды проиллюстрировать скриншотом экрана. Протокол оформить в формате ,doc документа с сохранением нумерации пунктов заданий лабораторной работы.
4. Ответы на контрольные вопросы

Вопросы для контроля:

1. Что такое командная оболочка? Как можно определить её тип?
2. Что такое файл?
3. Что такое каталог?
4. Что такое путь файла? Абсолютный и относительный путь?
5. Типы файлов, используемые в ОС Linux.

Лабораторная работа N 2
Знакомство с операционной системой LINUX
Способы хранения информации.
Команды управления данными. Изменение прав доступа к файлам.

Цель работы: получить базовые навыки по работе с операционной системой (ОС) Linux, ее командной оболочкой. Изучить понятия дерева каталогов, файла и типы файлов. Изучить основные команды по управлению и манипуляции данными.

2.1 Файловая система Linux.

Файлы в Linux играют ключевую роль. Данные пользователей хранятся в виде файлов. Доступ к периферийным устройствам компьютера, включая диски, CD-ROM, принтеры, терминалы, сетевые адаптеры и даже память обеспечивается с помощью файлов. Наконец, все программы, которые выполняются в системе, как системные, так и задачи пользователей, являются исполняемыми файлами. Информация на магнитных носителях (дисках, дискетах, лентах) хранится в виде файлов.
Файл - поименованная область данных па магнитном носителе. Как и во многих современных операционных системах, в LINUX файлы организованы в виде древовидной структуры (дерева), называемой *файловой системой,* которая может быть представлена как неориентированный древовидный граф, вершинам которого соответствуют файлы и каталоги. *Каталог (директория)* - элемент файловой системы, включающий в себя другие файлы и каталоги, т.е. каталог это вершина графа, имеющая больше чем одну инцидентную связь.
В Linux все доступное пользователям файловое пространство объединено в единое дерево каталогов, корневой каталог которого обозначается символом '/'. В DOS системах каждое устройство хранения данных имеет свой буквенный идентификатор (A: - Z:) и имеет свою файловую систему, корневой

каталог которой обозначается '/'. Однако это не означает, что в LINUX системе присутствует только одна файловая система. В большинстве случаев единое дерево, какое видит пользователь системы, составлено из нескольких отдельных файловых систем, которые могут иметь различную внутреннюю структуру, а файлы, принадлежащие этим системам, могут находиться на различных устройствах.

Путь файла - это совокупность каталогов, которые надо пройти, для того чтобы получить доступ к файлу.

Пути бывают *относительные* (начало пути находится в текущем каталоге) и *абсолютные* (началом пути является корневой каталог).

Типы файлов

В UNIX системах существует 6 типов файлов, различающихся по функциональному назначению и действиям операционной системы при выполнении тех или иных операций над ними:

- Обычный файл (regular file)
- Каталог (derectory)
- Специальный файл устройства (special device file)
- FIFO или именованный канал (named pipe)
- Связь (link)
- Сокет

Обычный файл

Обычный файл представляет собой наиболее общий тип файлов, содержащий данные в некотором формате. Для операционной системы такие файлы представляют собой просто последовательность байтов. Вся интерпретация содержимого файла производится прикладной программой, обрабатывающей файл. К этим файлам относятся текстовые файлы, бинарные данные, исполняемые программы и т. п.

Каталог

Каталог - это файл, содержащий имена находящихся в нем файлов, а также указатели на дополнительную информацию - *метаданные,* позволяющие операционной системе производить операции над этими файлами. С помощью каталогов формируется логическое дерево файловой системы. Каталоги определяют положение файла в дереве файловой системы, поскольку сам файл не содержит информации о своем местонахождении. Любая задача, имеющая право на чтение каталога, может прочесть его содержимое, но только ядро имеет право на запись в каталог.

Специальный файл устройства

Специальный файл устройства обеспечивает доступ к физическому устройству (диску, CD-ROM, floppy и т.д.). Доступ к устройствам осуществляется путем открытия, чтения и записи в специальный файл устройства. В UNIX системах различают *символьные* (character) и *блочные* (block) файлы устройств. Символьные файлы устройств используются для небуферизированного обмена данными с устройством. В противоположность этому блочные файлы позволяют производить обмен данными в виде пакетов фиксированной длины - *блоков.* Доступ к некоторым устройствам может осуществляться как через символьные, так и через блочные специальные файлы.

FIFO или именованный канал

FIFO или **именованный канал** - это файл, используемый для связи между процессами. FIFO впервые появились в System V UNIX, но большинство современных LINUX- систем поддерживают этот механизм.

Связь

Как уже говорилось, каталог содержит имена файлов и указатели на их метаданные. В то же время сами метаданные не содержат ни имени файла, ни указателя на это имя. Такая архитектура позволяет одному файлу иметь несколько имен в файловой системе. Имена жестко связаны с метаданными и,

соответственно, с данными файла, в то время как сам файл существует независимо от того, как его называют в файловой системе. Такая связь имени файла с его данными называется *жесткой связью* (hard link).

Сокеты

Сокеты предназначены для взаимодействия между процессами. Интерфейс сокетов часто используется для доступа к сети TCP/IP. В системах ветви BSD UNIX на базе сокетов реализована система межпроцессного взаимодействия, с помощью которой работают многие системные сервисы, например, система печати.

После регистрации в системе, LINUX делает текущим ваш домашний каталог и запускает программу оболочки. Если вы используете графический интерфейс (GUI), то после регистрации на экране будет отображаться рабочий стол. Если вам необходимо использовать командную оболочку, начните сеанс работы в режиме эмуляции терминала («переход»/система/ терминал».)

После перехода в окно терминала появится короткое сообщение, называемое сообщением дня. Оно может быть любого содержания или вообще не иметь его, это решает системный администратор. После этого на экран будет выдано приглашение (возможно такое)

[student@wpl student]$

С вами начинает общение командная оболочка ОС LINUX.

2.2 Команды управления данными

Приступим к изучению ряда полезных, а, иногда, просто необходимых команд управления операционной системой LINUX. Для использования этих команд следует перейти из графического режима ОС LINUX в

режим эмуляции терминала **« терминал»**.

Владельцы и права доступа к файлам.

Принадлежность файлов определенному владельцу - одно из основных средств защиты информации, хранящейся в файловой системе LINUX. Каждый файл имеет следующие категории прав доступа:

- Права владельца (u)
- Права группы(g)
- Права прочих пользователей(o)

Права владельца определяют, что может делать с файлом тот пользователь, которому этот файл непосредственно принадлежит. Права группы определяют, что могут делать с файлом члены группы, в которую входит владелец файла. И, наконец, права прочих пользователей определяют возможные действия тех пользователей, которые не принадлежат к предыдущим двум категориям.
Ниже перечислены возможные действия с файлом:

- Чтение (r)
- Запись (w)
- Выполнение (x)

Пользователь, имеющий право чтения, может просматривать содержимое файла. Пользователь, имеющий право записи, может редактировать этот файл. И, наконец, пользователь, имеющий право выполнения, может запускать программу, содержащуюся в этом файле.

Определение прав доступа

Определить права доступа к файлу или каталогу можно, используя команду **ls** с ключом **–l (эль)**. После ввода такой команды на экран будет выведена

подробная информация о каждом файле текущего каталога, например:

[student@wpl student]$**ls –l**<Enter>

1 2 3 4 5 6 7 8

drwxrwxr-- 5 student student 36 Dec 22 19:13 CONT
-rwxr-xr-- 1 andy group 4889 Aug 15 15:09 rtp.txt

Первый столбец этой информации будет содержать права, присвоенные каждому файлу по умолчанию системой. Первый символ указывает тип файла: d – каталог, - обычный файл. Набор прав состоит из девяти последующих символов (xxxxxxxxx). Первые три символа задают права на файл его владельцу (u), вторая тройка символов указывает на права группы (g), третья тройка – права прочих пользователей (o), зарегистрированных в системе LINUX. Символы, указывающие права пользователей на действия с файлом, перечислены в таблице 2

Таблица 1- Символьное обозначение прав

Символ	права	определение
r	Чтение	Пользователь имеет право просматривать содержимое файла
w	запись	Пользователь имеет право изменять содержимое файла
x	выполнение	Пользователь имеет право запускать файл, который должен представлять собой программу или сценарий. Если право на выполнение задано для каталога, это означает, что пользователь может обращаться к данному каталогу.

Изменение прав доступа к файлам и каталогам

Обычно для установки прав доступа к файлам и каталогам используется команда **chmod,** которая вызывается следующим образом:

[student@wpl student]$ **chmod** *[выражение] имя файла*

Устанавливаемые права доступа определяются значением *выражения*. При вызове команды **chmod** могут использоваться выражения двух типов: символьные и восьмеричные.

Символьные выражения.

В символьных выражениях, как и следует из их названия, для указания прав доступа используются символы (см. таблицу 2). Выражение такого типа имеет следующую структуру:

(категория пользователя) (действие) (права).

Категории пользователя могут быть следующие:

u -владелец файла,

g - группа,

o - прочие,

a - все.

Действие: "+" - добавление нового права доступа, "-" - отмена существующих прав, "=" - явное указание новых прав доступа. Например, команда

[student@wpl student]$**chmod u-w CONT**<Enter>

лишит владельца каталога CONT права **вносить изменения** в этот каталог.

А команда

[student@wpl student]$**chmod o+w CONT**<Enter>

добавит право изменять содержимое этого каталога всем прочим пользователям, зарегистрированным в системе LINUX.

Команда

[student@wpl student]$**chmod g=rwx rtp.txt** <Enter>

установит права на чтение, редактирование и выполнение файла rtp.txt всей группе пользователей, к которой принадлежит владелец этого файла.

Восьмеричные выражения.

При использовании восьмеричных выражений вы можете лишь явным способом задавать **полный список прав доступа к файлу или каталогу**. Восьмеричное число представляет новые права для всех категорий пользователей (владельца, группы, прочих). Праву на чтение соответствует восьмиричное число 4 (или двоичное 100), на запись - восьмиричное число 2 (или двоичное 010), на выполнение – 1 (или двоичное 001). Таким образом, значение, определяющее права доступа конкретной категории пользователей к данному файлу, лежит в диапазоне от 0 до 7. Число, состоящее из трех цифр, лежащих в диапазоне от 0 до 7, определяет права для владельца файла, группы и остальных пользователей. Например, команда

[student@wpl student]$ **chmod 751 ммм**

присваивает файлу ммм следующие права доступа:

Первая цифра -7 (в двоичной системе это 111) владельцу файла разрешает читать(r), редактировать(w) и исполнять(x) файл ммм.

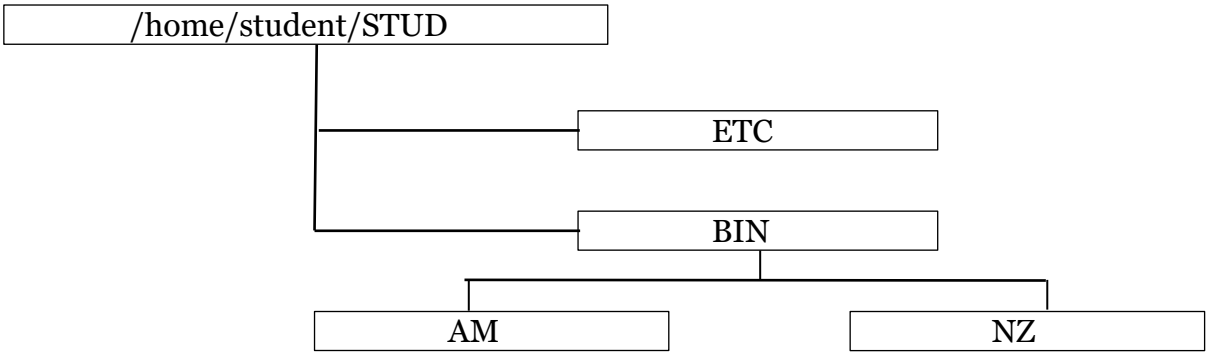
Вторая цифра -5 в двоичной системе представляет набор 101, т.е. членам группы разрешено файл читать(r) и исполнять(x).

Прочим пользователям (третья цифра- 1) разрешено только 001, т.е. только файл исполнять.

Задание для лабораторной работы №2

Вариант 1.

Создать структуру. Назначить разрешения согласно таблице

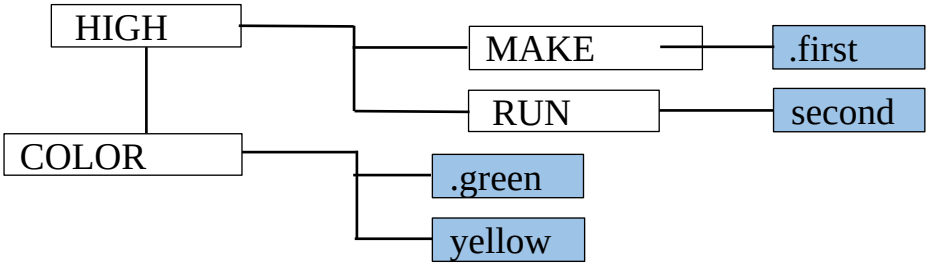


Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
ETC	р	з	р	р	з	р	з	з	з	символьный
BIN	р	р	р	з	з	р	з	з	з	числовой
AM	р	р	р	р	з	з	з	р	р	числовой
NZ	р	з	р	з	р	з	р	з	з	символьный

Применив маску, в каталоге **/etc** найти файлы, оканчивающиеся на **“conf”.** Записать их в подкаталог **ETC**. В каталоге **/bin** определить файлы, состоящие из семи символов. Те из них, что начинаются с **a, b, … m,** записать в **AM,** остальные в **NZ.**

Вариант 2.

Создать структуру. Назначить разрешения согласно таблице.



Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
HIGH	р	з	р	р	з	р	з	з	з	символьный
.first	р	р	р	з	р	з	р	р	з	числовой
COLOR	р	р	р	з	з	р	з	з	з	числовой
second	р	р	з	з	з	з	р	р	р	символьный
MAKE	р	р	р	р	з	з	з	р	р	числовой
RUN	р	з	р	з	р	з	р	з	з	символьный
.green	з	з	з	р	з	з	з	з	р	числовой
yellow	р	р	з	з	р	з	р	з	з	символьный

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.

В файл **first** добавить две команды:

1) вывести оглавление директорий HIGH и COLOR в рекурсивном виде;
2) создать копию каталога RUN в COLOR.

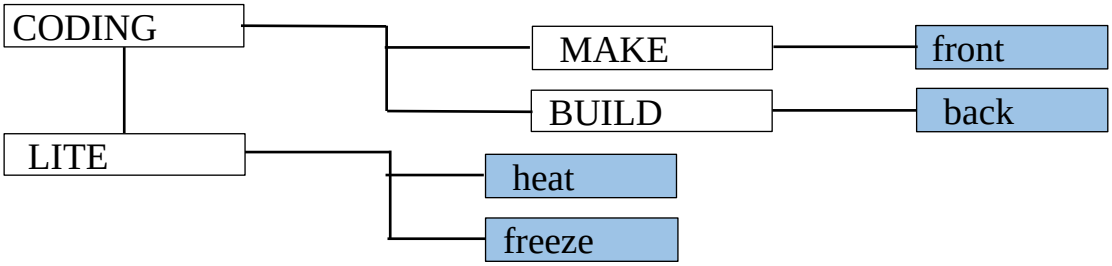
Вариант 3.

Создать два подкаталога, например, user1 и user2. В домашнем каталоге user1 создать директорию с файлом. User1 наделить правами чтения и редактирования файла, user2 читать и запускать файл. В файл добавить

команду, выводящую оглавление текущего каталога. Для изменения прав использовать утилиту управления списками контроля доступа.

Вариант 4.

Создать структуру. Назначить разрешения согласно таблице.



Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
CODING	р	р	р	р	з	р	з	з	з	числовой
MAKE	з	р	з	з	р	з	р	р	з	числовой
BUILD	р	р	р	з	з	р	з	з	з	числовой
LITE	р	р	р	з	з	з	р	р	р	символьный
front	р	р	р	р	з	з	з	р	р	числовой
back	р	з	р	з	р	з	р	з	з	символьный
heat	з	з	з	р	з	з	з	з	р	символьный
freeze	р	р	р	з	р	з	р	з	з	числовой

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.

В файл **freeze** добавить команды:

1) переименовать каталог BUILD в COMPILE;
2) из каталога /usr/share/man/man1 выдать информацию о файлах, содержащих в имени подстроку **bug** или **play.**

Вариант 5.

В домашнем каталоге создать директорию, а в ней произвольное количество файлов и подкаталогов. С помощью утилиты управления списками контроля доступа назначить сразу всем объектам следующие права:

владелец: чтение+выполнение;

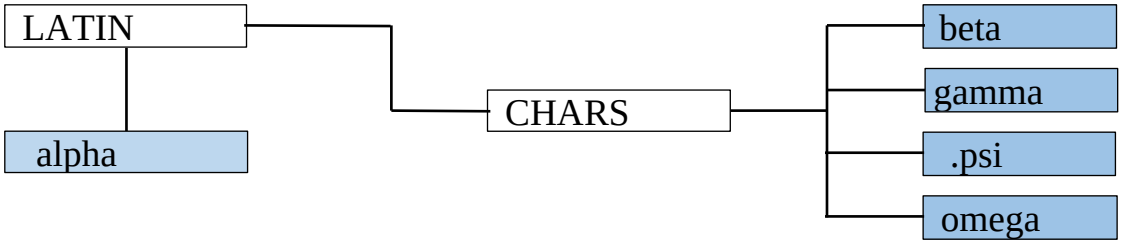
группа: прав нет;

остальные: только чтение.

Применить как символьную, так и числовую запись.

Вариант 6.

Создать структуру. Назначить разрешения согласно таблице.



Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
LATIN	р	р	р	р	з	р	з	з	з	символьный
CHARS	з	р	р	з	р	з	р	р	з	числовой
gamma	р	р	р	з	з	р	з	з	з	числовой
alpha	з	з	з	з	з	з	р	р	р	символьный
beta	р	р	р	р	р	з	з	р	з	числовой
.psi	р	з	р	з	р	з	р	з	з	числовой
omega	з	з	з	р	з	з	з	з	р	символьный

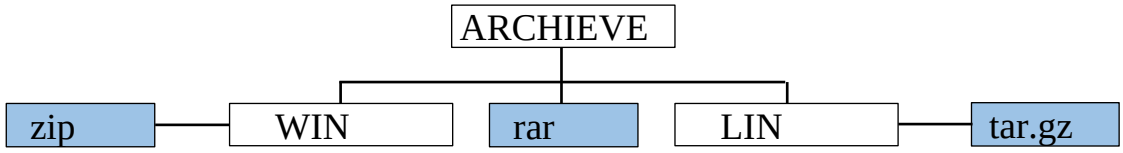
До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.

В файл **gamma** добавить команды:

1) создать три копии данного файла в текущей директории;
2) вывести в одной строке текущую дату, в другой время (в виде 02.10.2022 и 14:45:07).

Вариант 7.

Создать структуру. Назначить разрешения согласно таблице.



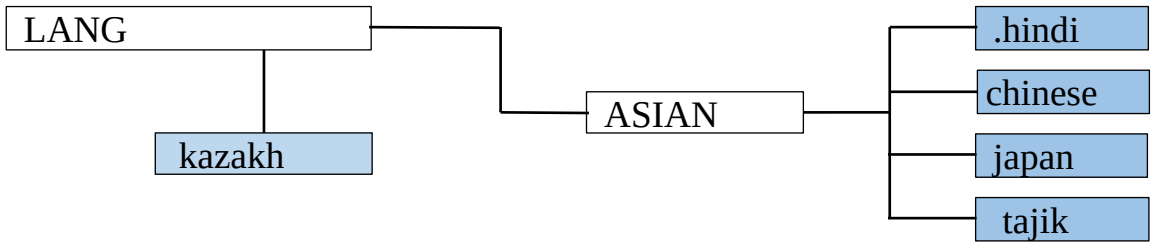
Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
zip	р	р	р	з	р	з	з	р	р	символьный
rar	з	р	з	з	р	р	р	р	з	числовой
tar.gz	р	р	з	р	з	р	з	р	з	символьный
WIN	р	з	р	р	р	з	з	р	з	числовой
LIN	з	з	з	з	р	з	р	з	з	символьный
ARCHIEVE	р	з	р	р	з	з	з	з	р	числовой

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.
В файл **zip** добавить команды:

1) вывести значение текущего рабочего каталога;
2) напечатать содержимое файла /etc/os-release с нумерацией строк.

Вариант 8.

Создать структуру. Назначить разрешения согласно таблице.



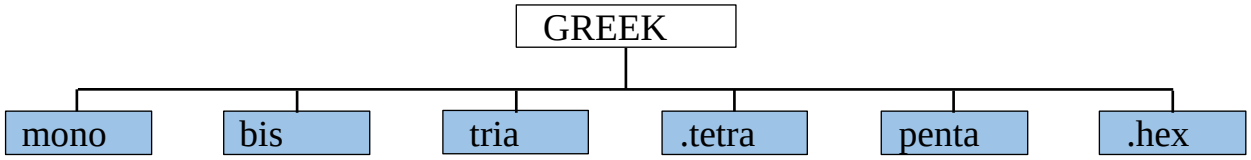
Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
kazakh	р	р	р	р	з	р	з	з	з	числовой
.hindi	з	р	з	з	р	з	р	р	з	числовой
chinese	р	з	з	з	р	з	р	р	з	числовой
japan	з	з	з	з	з	з	р	р	р	символьный
tajik	р	р	з	р	р	з	з	р	з	числовой
LANG	з	р	р	з	р	з	р	з	з	числовой
ASIAN	р	р	р	р	з	з	з	з	р	символьный

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.
В файл **kazakh** добавить команды:

1) вывести на экран текст из файла tajik с построчной нумерацией;
2) переместить файлы hindi, chinese, japan в директорию LANG.

Вариант 9.

Создать структуру. Назначить разрешения согласно таблице.



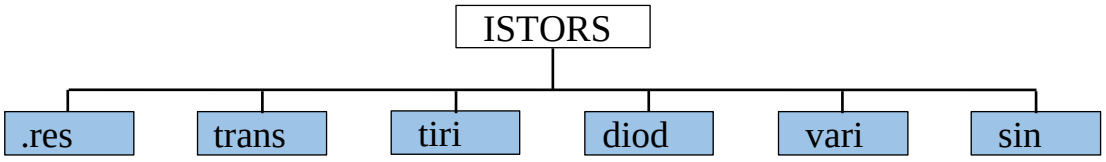
Файл/	Права доступа (р – разрешить, з - запретить)	Способ записи

	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
mono	р	р	р	р	з	з	з	з	з	символьный
bis	з	р	з	з	р	з	з	р	р	числовойй
tria	р	з	з	з	р	з	р	р	з	символьный
.tetra	з	з	з	р	з	р	з	р	з	чмсловой
penta	р	р	з	р	р	з	з	р	з	символьный
.hex	з	р	р	з	р	з	р	з	з	символьный
GREEK	р	р	р	р	з	з	з	з	р	числовой

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.
В файл **mono** добавить команды:

1) вывести оглавление каталога GREEK с сортировкой по возрастанию размера файла;
2) удалить из GREEK только те файлы, в которых встречается хотя бы одна буква **е** без вывода на экран запроса на подтверждение удаления.

Вариант 10.
Создать структуру. Назначить разрешения согласно таблице.



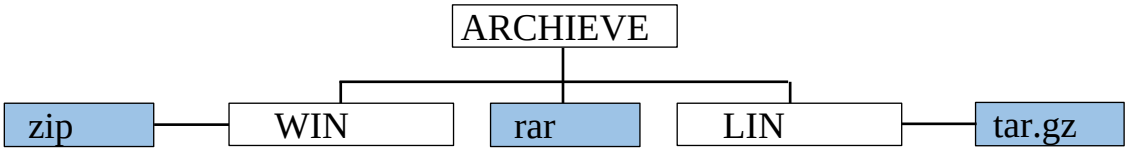
Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
.res	р	з	р	р	з	з	р	з	з	символьный
trans	р	р	з	з	р	з	з	р	р	символьный
tiri	р	з	з	з	р	з	р	р	з	числовой
diod	р	р	р	р	з	р	з	р	з	символьный
vari	р	з	з	р	р	з	з	р	з	числовой
sin	р	р	р	з	р	з	р	з	з	символьный
ISTORS	р	з	р	р	з	з	з	з	р	числовой

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.
В файл **diod** добавить команды:

1) создать копию каталога ISTORS в директории /tmp;
2) лишить всех пользователей всех прав на этот каталог.

Вариант 11.

Создать структуру. Назначить разрешения согласно таблице.



Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
zip	р	р	р	з	р	з	з	р	р	символьный
rar	з	р	з	з	р	р	р	р	з	числовой
tar.gz	р	р	з	р	з	р	з	р	з	символьный
WIN	р	з	р	р	р	з	з	р	з	числовой
LIN	з	з	з	з	р	з	р	з	з	символьный
ARCHIEVE	р	з	р	р	з	з	з	з	р	числовой

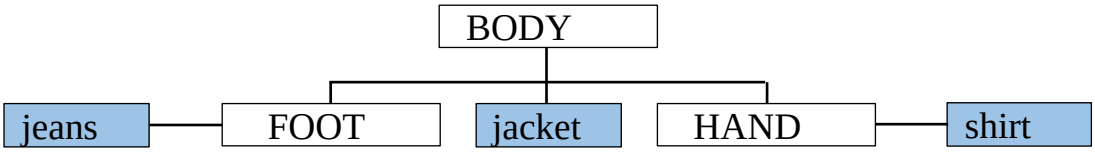
До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.

В файл **zip** добавить команды:

3) вывести значение текущего рабочего каталога;
4) напечатать содержимое файла /etc/os-release с нумерацией строк.

Вариант 12.

Создать структуру. Назначить разрешения согласно таблице.



Файл/ каталог	Права доступа (р – разрешить, з - запретить)									Способ записи
	владелец			группа			остальные			
	чт	зап	вып	чт	зап	вып	чт	зап	вып	
jeans	р	з	р	з	р	з	з	з	з	символьный
jacket	р	р	р	з	р	р	р	з	з	числовой
shirt	р	з	з	з	з	р	з	р	з	числовой
FOOT	з	р	р	р	р	з	з	р	з	символьный
HAND	з	з	р	з	р	з	р	з	з	числовой
BODY	р	з	р	р	з	з	з	з	р	числовой

До/после изменения прав выполнить проверку наличия/отсутствия какого-либо из прав (например, выдать оглавление каталога/содержимое файла в случае изменения права на чтение). Таким образом проверить каждый файл/каталог.

В файл **jacket** добавить команды:

1) создать пустой каталог BODY в директории /tmp;
2) скопировать в него все три файла.

Требование к отчету о лабораторной работе

Отчет по лабораторной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель лабораторной работы

3 Полный протокол выполнения задания лабораторной работы с пояснением каждой команды. Результаты работы каждой команды проиллюстрировать скриншотом экрана. Протокол оформить в формате ,doc документа Ответы на контрольные вопросы

Вопросы для контроля:

1. Что такое командная оболочка? Как можно определить её тип?
2. Что такое файл?
3. Что такое каталог?
4. Как определить права доступа на файл или каталог?
5. Как изменить права доступа к файлу или каталогу?
6. Чем отличается символьный способ изменения прав от способа задания прав в восьмиричном формате?

Лабораторная работа N 3
Списки команд
Каналы ввода-вывода.
Перенаправление каналов

Цель работы: Изучение стандартных потоков ввода – вывода. Перенаправление каналов.

3.2. Списки команд

Часто бывает необходимо выполнить несколько команд последовательно одну за другой, т.е. *списком.* Для этого команды объединяются символом "**;**"

[student@wpl student]$**команда1 ; команда2 ;** и т.д. <Enter>

Здесь **команда1 ; команда2 ;** - это любые команды, из уже изученных в теме1, а также команды, которые предстоит изучит в последующих темах нашего практикума.

Списки также применяются в тех случаях, когда выполнение команды должно зависеть от статуса завершения предыдущей команды. При этом используются логические операторы:

[student@wpl student]$**команда1 && команда2** <Enter>
[student@wpl student]$ **команда1 || команда2** <Enter>

Логические операции используются так же, как и в языке Си.

В первом случае (операция &&) команды объединены оператором конъюнкции. При этом комнада2 выполняется только в том случае, когда статус завершения команды1 равен 0(т.е. команда завершена успешно). Во втором случае (операция ||) команды связаны между собой операцией дезъюнкции, и команда2 выполняется только в случае, если команда1 возвращает значение не равное 0(т.е. команда1 закончила свою работу с ошибкой). Например:

[student@wpl student]$**mkdir docs && cd docs** <Enter>

В этом списке команд делается попытка создать каталог с именем docs (команда **mkdir**) и сделать этот каталог текущим (команда **cd**). Команда **cd** будет исполнена только в том случае, если команда создания каталога пройдет успешно.

3.3. Каналы ввода-вывода.

Часто при обработке информации в UNIX- системах выходные данные одной команды поступают на вход другой. Такой способ взаимодействия команд называется *конвейером.* Для организации конвейера используются каналы:

[student@wpl student]$**команда1 | команда2 |** и т.д.

Внимание: Обратите внимание, что при организации конвейера используется операция "| ", а не "||" как в случае списка команд.

Канал, задаваемый символом **“|”,** соединяет выход команды1 со входом команды2 и т.д. Используемые команды могут быть как простыми, так и сложными. Например:

[student@wpl student]$**cat /etc/passwd | *lpr***

или

[student@wpl student**]$tar -cf -./fff. tar |{cd \~/; tar -xf -}.**

(Назначение этих команд см. в электронном справочнике **man**).

В случае использования сложных команд применяется группировка –

{ команды } или (команды). В первом случае команды, заключенные между фигурными скобками, исполняются текущей командной оболочкой. Если используются простые скобки, то выполнение заключенных между ними команд возлагается на новый экземпляр оболочки, что иногда бывает необходимым, но чревато перерасходом ресурсов системы.

3.4. Перенаправление каналов

Стандартным каналом вывода любой информации является вывод на экран монитора.

Перенаправление вывода производится с помощью оператора >, например:

[student@wpl student]$ **команда > файл** <Enter>

Приведенный оператор перенаправления вывода (результатов работа команды) приведет к созданию (если такой файл не существует) или перезаписи (в противном случае) указанного файла. Если же надо информацию добавить в уже существующий файл, то используется оператор " >>*″:*

[student@wpl student]$**команда >> файл** <Enter>

Кроме перенаправления стандартного вывода, иногда приходится также перенаправить стандартный канал вывода сообщения об ошибке при выполнении команды. В таком случае используется оператор: '**2>**', т.е. к оператору вывода

добавляется цифра 2, что соответствует 2-му стандартному потоку, т.е. потоку ошибок. Возможно также объединение этих двух потоков в один. Например, для того чтобы вывести результаты компиляции программы, написанной на языке Си:

[student@wpl student]$**gcc file.c -Wall -ansi -o file 2>&1 > results.**

Здесь стандартный вывод (дескриптор 1) и стандартный вывод ошибки (дескриптор 2) объединяются и перенаправляются в указанный файл **results**.

Стандартным каналом ввода является ввод с клавиатуры.

Если требуемые вам данные лежат в файле, их можно оттуда взять, используя оператор перенаправления "<":

[student@wpl student]$**команда < файл** <Enter>

Если в качестве файла будет использоваться все-таки ввод с клавиатуры, то можно использовать модификацию этого оператора:

[student@wpl student]$**команда << разделитель** <Enter>

что будет обозначать, что в команду пойдут данные с клавиатуры, а признаком конца данных будет указанный разделитель.

Операторы перенаправления ввода и вывода могут присутствовать одновременно в одной команде.

Например, создать файл fff, содержащий строку *Привет мир!!!* можно так:

[student@wpl student]$**cat > fff << END**<Enter>

>**Привет мир !!!**
>**END**<Enter>

Убедимся, что файл создан и содержит именно указанный текст.

[student@wpl student]\$**ls ; cat fff** <Enter>

fff
Привет мир !!!

[student@wpl student]$

Задание для лабораторной работы №3

Общее задание для всех:

1. Используя перенаправление канала вывода вывести содержимое файла, созданного в лабораторной работе №1 в файл с именем **envs**

2. Убедитесь, что первое задание вами выполнено верно.
Для этого выведите содержимое файла **envs** на экран и сравните с исходным текстом.

3. Вывести на экран содержимое файла /etc/passwd. Проанализировать полученную на экране информацию.

4. Используя канал «конвейер» и перенаправление вывода выполнить следующие действия **списком команд**:

- перейти в домашний каталог;
- выдать содержимое файла /etc/passwd в файл passwd.orig, предварительно отсортировав его по имени пользователей.

(для проведения сортировки файла используйте команду **sort**);

5.Проанализируйте смысл файла **passwd.orig**. (см. смысл файла /etc/passwd в лаб. работе №1)

6. Добавить в файл **passwd.orig** информацию о новом пользователе согласно формату записи файла /etc/passwd (все поля обязательно должны быть заполнены), используя перенаправление ввода с разделителем и перенаправление вывода. Убедитесь, что добавление записи прошло успешно.

7. Создать файл a1 с помощью команды cat; ввести в файл текст из 6-ти строк (строка начинается с цифры- порядкового номера строки).

8. Покажите содержимое созданного файла a1 на экране.

Лабораторная работа N 4
Способы хранения информации.
Команды управления данными

4.1 Команды управления данными

Еще несколько полезных, а, иногда, просто необходимых команд управления файлами в операционной системе LINUX. Для использования этих команд следует перейти из графического режима ОС LINUX в режим эмуляции терминала « терминал».

Команда head

Команда head записывает в стандартный поток вывода первые несколько строк каждого из заданных файлов или первые несколько строк из стандартного потока ввода. Если в команде head не указаны флаги, то по умолчанию выводятся первые 10 строк.

Синтаксис у команды head следующий:

$ head опции файл

Здесь:

- **Опции** — это параметр, который позволяет настраивать работу команды таким образом, чтобы результат соответствовал конкретным потребностям пользователя.
- **Файл** — это имя документа (или имена документов, если их несколько). Если это значение не задано либо вместо него стоит знак «-», команда будет брать данные из стандартного вывода.

Чаще всего к команде head применяются такие опции:

- **-c (--bytes)** — позволяет задавать количество текста не в строках, а в байтах. При записи в виде --bytes=[-]NUM выводит на экран все содержимое файла, кроме NUM байт, расположенных в конце документа.
- **-n (--lines)** — показывает заданное количество строк вместо 10, которые выводятся по умолчанию. Если записать эту опцию в виде --lines=[-]NUM, будет показан весь текст кроме последних NUM строк.
- **-q (--quiet, --silent)** — выводит только текст, не добавляя к нему название файла.

- **-v (--verbose)** — перед текстом выводит название файла.
- **-z (--zero-terminated)** — символы перехода на новую строку заменяет символами завершения строк.

Примеры использования head

Самый простой способ использования команды head — с указанием имени файла, но без опций. В таком случае будут выведены на экран первые 10 строк.

[student@wpl student]$**head file-name.txt**

Если нужно единовременно получить вывод с нескольких файлов, с этим тоже не возникнет проблем. Достаточно перечислить названия, разделяя их пробелом:

[student@wpl student]$**head file-name1.txt file-name2.txt**

Чтобы название файла выводилось даже в том случае, когда команде задан только один документ, следует воспользоваться опцией -v:

[student@wpl student]$**head -v file-name.txt**

Если десяти строк, по умолчанию выводимых командой, окажется слишком мало или слишком много, ничто не мешает изменить их количество вручную. Для этой цели служит опция -n:

[student@wpl student]$**head -n 4 file-name.txt**

Команда head , которая выводит начальные строки файла, - это в некотором роде антагонист команды tail (она печатает в терминале последние строки).

Команда tail

`tail` — отличная команда Linux, используемая для вывода N-го количества последних строк файла. Обычно она показывает в стандартном выводе 10 последних строк из файла. Если мы запускаем её для не одного имени файла, данные из каждого файла обрабатываются по имени файла с заголовком.

Синтаксис команды tail

```
tail [OPTION]... [FILE]...
```

Параметр ` -v` в команде tail

При использовании этой опции данным из указанного файла всегда предшествует имя файла.

[student@wpl student]$**tail -v state.txt**

```
administrator@GFG19566-LAPTOP:~$ tail -v state.txt
==> state.txt <==
Odisha
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh
Uttarakhand
WestBengal
administrator@GFG19566-LAPTOP:~$
```

Параметр `-n` num в команде tail

Печатает последние 'num' строк вместо последних 10 строк. **num** обязательно должен быть указан в команде, иначе выводится ошибка. Эту команду можно также записать без символа 'n', но знак '-' обязателен.

```
administrator@GFG19566-LAPTOP:~$ tail -n 3 state.txt
Uttar Pradesh
Uttarakhand
West Bengal
administrator@GFG19566-LAPTOP:~$ tail -3 state.txt
Uttar Pradesh
Uttarakhand
West Bengal
administrator@GFG19566-LAPTOP:~$
```

Утилитаfind

`find` — одна из самых полезных команд Linux для тех, кто работает с сотнями и тысячами файлов на своём компьютере. Например, для системных администраторов. Как можно догадаться из названия, она помогает найти нужные файлы и каталоги, причём не только по их именам.

Команда find имеет определённую структуру:

find [каталог] [параметры] [критерий поиска] [паттерн]

Разберём её по частям:

find — команда запуска поиска.

Каталог — это путь к папке, в которой мы будем что-то искать. Если не задать его вручную, то по умолчанию используется /home/имя_учётной_записи, где хранятся файлы, созданные пользователем.

Параметры — дополнительный критерий поиска. Например, можно искать только файлы или только каталоги.

Критерий поиска — указание на особенности файлов, по которым мы будем их искать: имя, дата создания или расширение.

Паттерн — значение, по которому будут отбираться файлы: их дата изменения, права доступа и так далее.

Рассмотрим варианты применения утилиты find. Начнём с простого поиска и затем перейдём к настройке критериев поиска и паттернов.

Команда find без дополнительных параметров

Если вы введёте команду find без дополнительных параметров, то получите список всех файлов, хранящихся в каталогах, начиная с текущего (об этом говорит «.», поставленная перед /).

Поиск файлов в определённом каталоге

Откроем папку Документы и посмотрим, какие файлы в ней есть. Для этого используем команду:

[student@wpl student]$ **find ./Документы**

Поиск по имени: find name

Чтобы найти файл по имени, нужно к find добавить критерий поиска -name. Название не обязательно писать целиком. Будет достаточно нескольких букв и символа * после них. Например, Озе*, вместо Озеро. Так find найдёт в текущем каталоге все файлы, содержащие такой набор букв в названии:

[student@wpl student]$**find . -name ”Озе*”**

Поиск по расширению

Для того чтобы найти файлы с определённым расширением, к строке поиска добавляется его наименование в двойных кавычках. Например, изображения в формате JPG, можно найти так:

[student@wpl student]$**find -name "*.jpg"**

Для поиска можно указать сразу несколько форматов, например .jpg и .webp. Для этого используется команда -o, выполняющая функцию логического оператора ИЛИ:

[student@wpl student]$**find -name "*.jpg" -o -name "*.webp"**

Поиск по типу файла: find file

`find` по умолчанию ищет как файлы, так и каталоги. Это не всегда удобно. Чтобы провести поиск только в одной из этих категорий, используйте параметр -type с модификатором:

- d — для каталогов;
- f — для файлов.

Попробуем найти каталог .var. Явно укажем, что ищем именно папку, а не файл с таким именем:

[student@wpl student]$**find . -type d -name ".var"**

Поиск с исключением

Если есть несколько форматов файлов, например изображений с озером, можно исключить часть из них. Вернёмся к изображениям со словом «Озеро» в названии и найдём все, кроме тех, что имеют формат .webp.

Для этого используем уже знакомый логический оператор -not, который укажем перед параметром для исключения:

[student@wpl student]$**find . -name "Озеро*" -not -name "*.webp"**

Поиск в нескольких каталогах

Иногда файлы, которые требуется найти, находятся не в одном каталоге, а в нескольких. Чтобы не писать команду поиска дважды, можно перечислить папки через пробел.

Найдём все файлы с расширениями .jpg и .docx, которые находятся в каталогах Документы и Изображения. Имена папок указываем после find ., не забывая про символ / перед ними:

[student@wpl student]$**find ./Документы/Изображения –type f –name ”*.jpg” –o –name ”*.docx”**

Поиск файлов, принадлежащих пользователю

Чтобы найти файлы, принадлежащие конкретному пользователю, можно применить одну из двух команд:

[student@wpl student]$**find -user имя_пользователя**

или

[student@wpl student]$**find ./Документ -user имя_пользователя**

Поиск пустых файлов и каталогов

Чтобы найти и вывести информацию о пустых файлах и папках, используется параметр -empty. Найдём пустые папки:

[student@wpl student]$ **find . -type d –empty**

Команда grep

grep — это команда для поиска внутри текстовых файлов. Она, так же как и команда find, есть во всех популярных Linux-дистрибутивах. Рассмотрим сначала её простое использование, а потом научимся более сложным вещам.

Фишка грепа в возможности поиска совпадений внутри множества файлов. Для этого используется параметр -r — рекурсивный поиск в каталогах. Например, мы хотим найти вхождения подстроки «Content to export» внутри всех файлов нашего проекта. Выглядеть такая команда будет так:

```
grep -rn --include="*.php" "Content to export" .
```

Тут мы применили целый набор параметров для более точного поиска вхождений:

- -r — рекурсивный поиск в каталогах;
- -n — выводить номера строк, на которых было найдено вхождение;
- --include=”*.php” — искать только в файлах с расширением php. Если этот параметр убрать — поиск будет осуществляться по всем файлам указанной директории;
- "Content to export" — вхождение, которое мы ищем;
- **.** — где мы ищем. В нашем случае внутри текущей директории и вниз.

grep позволяет искать и более сложные вещи, чем одно вхождение, например, мы можем найти несколько разных вхождений. Это делается с помощью специальной конструкции: 'шаблон 1\|шаблон 2'. Здесь обратный слэш экранирует оператор пайп — «|», поэтому шаблон разделяется на два шаблона: шаблон 1 и шаблон 2.

Вот как может выглядеть поиск нескольких разных вхождений:

```
grep -rn --include="*.php" "Content to export\|Upload content" .
```

grep — утилита мощная, но старая и сегодня скорость её работы и удобства оставляют желать лучшего. Однако она есть почти во всех Linux-дистрибутивах и опыт её использования может вас сильно выручить. Более современный и производительный вариант grep — это ripgrep. Работу с ним вы можете рассмотреть самостоятельно.

ripgrep — современный вариант grep

Команда cut

Команда cut используется, если нужно вырезать часть текста — при этом он может находиться в файле либо быть напечатанным через стандартный ввод (клавиатуру). В Unix-системах эта команда удаляет секции текста, которые были обозначены при помощи байтов, символов или полей, разделенных знаками "-" и ":". Работу cut обеспечивает одноименная утилита.

Написание команды cut выглядит следующим образом:

```
cut опции путь_к_файлу
```

Использовать файл не обязательно. Если на месте его названия поставить прочерк - либо не указать ничего, команда возьмет текст из стандартного ввода. При необходимости можно указывать больше файлов, чем один.

Параметры cut

Список опций, которые позволяют управлять поведением команды:

- **-b (--bytes=LIST)** — номер байта, набор или диапазон байтов, подлежащих вырезанию.
- **-c (--characters=LIST)** — символ, который следует вырезать. Также можно указывать набор либо диапазон символов.

- **-d (--delimiter=DELIM)** — с помощью этой опции пользователь устанавливает свой разделитель вместо стандартного TAB.
- **-f (--fields=LIST)** — перечень полей для вырезания.
- **-s (--only-delimited)** — если была применена эта опция, cut не выводит строки, где нет разделителя.
- **--complement** — задает байты, символы или поля, которые останутся в файле или тексте из стандартного ввода. Все остальное будет вырезано.
- **--output-delimiter=STRING** — по умолчанию выходной разделитель соответствует входному. Эта опция позволяет задать другой выходной разделитель.
- **-z, --zero-terminated** — вместо символа новой строки разделителем будет NULL.

Примеры использования cut в Linux

Прежде всего создадим файл example.txt и поместим его в домашнюю директорию. В теле документа пропишем текст:

Winter: white: snow: frost
Spring: green: grass: warm
Summer: colorful: blossom: hot
Autumn: yellow: leaves: cool

Варианты использования команды, где нужно вырезать символы из текста, который находится в файле. Стандартная запись в терминале выглядит так:

[student@wpl student]$ **cut -b 1,9 example.txt**

Примерно так же работает вырезание символов в заданном диапазоне. Диапазон - это два числа, написанные через дефис:

[student@wpl student]$ **cut -b 12-20 example.txt**

Команду cut можно использовать саму по себе, но не возбраняется сочетать с другими командами. Чаще всего используется sort. Попробуем вырезать первые 4 символа и расположить строки в алфавитном порядке:

[student@wpl student]$ **cut -b 1-7 example.txt | sort**

Чтобы вырезать из текста, напечатанного через стандартный ввод, первый символ, команда cut должна иметь вид:

[student@wpl student]$ **echo "The sky was yellow as brass." | cut -b 1**

Команда sort

Это утилита для вывода текстовых строк в определенном порядке. Проще говоря, для сортировки. Ее можно использовать для сортировки текста из одного или нескольких файлов или с помощью нее может быть выполнена сортировка вывода linux для какой-либо команды. Это может быть полезно во многих случаях. Например, отсортировать файлы по размеру в выводе команды du или собрать частотность использования команд из истории.

Рассмотрим возможности команды sort Linux, ее опции и разберем несколько примеров использования.

Рассмотрим общий синтаксис команды:

```
$ sort опции файл
```

Или

```
$ команда | sort опции
```

Опции

Теперь рассмотрим основные опции утилиты sort.

- **-b** - не учитывать пробелы
- **-d** - использовать для сортировки только буквы и цифры
- **-i** - сортировать только по ASCII символам
- **-n** - сортировка строк по числовому значению
- **-r** - сортировать в обратном порядке
- **-c** – проверить, был ли отсортирован файл
- **-o** - вывести результат в файл
- **-u** - игнорировать повторяющиеся строки
- **-m** - объединение ранее отсортированных файлов
- **-k** - указать поле, по которому нужно сортировать строки, если не задано, сортировка выполняется по всей строке.
- **-f** - использовать в качестве разделителя полей ваш символ вместо пробела.

Примеры использования sort

Давайте сначала создадим файл с несколькими строками, на котором и будем проверять возможности утилиты.

[student@wpl student**]$ nano test.txt**

Сделайте его содержимым следующие строки

computer
mouse
LAPTOP
data
RedHat
laptop
debian
lapto

1. Сортировка

Теперь давайте выполним сортировку строк в нашем файле:

[student@wpl student]$ **sort test.txt**

```
computer
data
debian
laptop
laptop
LAPTOP
mouse
RedHat
```

Вот несколько принципов, по которым команда sort сортирует строки:

- Строки с цифрами размещаются выше других строк
- Строки, начинающиеся с букв нижнего регистра размещаются выше
- Сортировка выполняется в соответствии алфавиту
- Строки сначала сортируются по алфавиту, а уже вторично по другим правилам.

2. Обратная сортировка

Отсортируем файл в обратном порядке:

[student@wpl student]$ **sort -r test.txt**

```
RedHat
mouse
LAPTOP
laptop
laptop
debian
data
computer
```

3. Сортировка по колонке

Отсортируем вывод команды ls по девятой колонке, то есть по имени файла или папки. Колонку укажем опцией -k:

[student@wpl student]$ **ls -l | sort -k9**

```
drwxr-xr-x 6 user user 4096 дек 6 14:29 Android
drwx------ 3 user user 4096 янв 14 22:18 Desktop
drwxr-xr-x 12 user user 4096 янв 14 21:49 Documents
drwx------ 5 user user 12288 янв 15 14:59 Downloads
drwxr-xr-x 7 user user 4096 янв 13 11:42 Lightworks
```

Сортировка вывода Linux выполняется так же просто, как и строк из файла.

4. Сортировка по номеру

Отсортируем вывод команды ls по второй колонке. Для сортировки по числовому значению используется опция -n:

[student@wpl student]$ **ls -l | sort -nk2**

```
drwx------ 5 user user 12288 янв 15 14:59 Downloads
drwxr-xr-x 6 user user 4096 дек 6 14:29 Android
drwxr-xr-x 7 user user 4096 июн 10 2015 Sources
drwxr-xr-x 7 user user 4096 окт 31 15:08 VirtualBox
drwxr-xr-x 7 user user 4096 янв 13 11:42 Lightworks
drwxr-xr-x 8 user user 12288 янв 11 12:33 Pictures
```

5. Удаление дубликатов

Команда sort позволяет не только сортировать строки, но и удалять дубликаты. Для этого есть опция -u:

[student@wpl student**]$ sort -u test.txt**

```
computer
data
debian
laptop
```

```
LAPTOP
mouse
RedHat
```

Теперь строчка laptop не повторяется.

6. Сортировка по нескольким полям

Мы можем сортировать данные по нескольким полям. Например, отсортируем вывод ls по второму первично и вторично девятому полям:

[student@wpl student]$ **`ls -l | sort -t "," -nk2,5 -k9`**

```
drwxr-xr-x 2 user user 4096 дек 6 14:32 Links
drwxr-xr-x 2 user user 4096 янв 13 10:43 tmp
drwx------ 3 user user 4096 янв 14 22:18 Desktop
drwxr-xr-x 3 user user 4096 мар 28 2015 Журналы
drwx------ 4 user user 12288 янв 15 15:42 Загрузки
```

Вот и все. Мы немного приоткрыли занавесу над возможностями сортировки строк linux с помощью команды sort.

Задания к лабораторной работе №4

Общее задание для всех

1. Создать новую рабочую директорию для выполнения заданий лабораторной работы №4.

2. **Создать в новой директории файл test.txt**
   Сделайте его содержимым следующие строки

   1 Computer
   2 Mouse
   3 LAPTOP
   4 Data
   5 RedHat
   6 Laptop
   7 Debian
   8 lapto

3. Создать в новой директории файл a2 с помощью команды touch;
   в редакторе ввести в файл текст из 6-ти строк и не менее двух слов в каждой строке. Каждую строку начинать с цифры

4. Убедиться, что файлы созданы; просмотреть их содержимое.

5. Результат вывода команды ls –l занести в файл f3.

6. Первые четыре строки файла **test.txt** занести в файл a2.

7. Добавить в файл f3 две последние строки файла **test.txt**.
   Просмотреть содержимое файла f3.

8. Отсортировать файл a2 по 2-му столбцу. Результат сортировки, используя опцию команды сортировки, вывести в файле a2_s2.
   Убедиться, что файл a2_s2 создан; просмотреть его содержимое.

9. Создать файл mix, содержащий 1-ю и 2-ю строки файла a2, 3-ю строку файла **test.txt**, 4 и 5-ю строки файла f3.

10. С помощью команды grep найти в файлах новой директории строки, содержащие цифру “3”. Проанализировать информацию на экране. Повторив команду, результат выполнения команды занести в файл a_g.

11. Используя команды grep, cut и sort, получить список имѐн тех файлов в каталоге /etc/, в тексте которых содержится запись ip-адреса.
    Список имен файлов не должен иметь повторений.
    Результат занести в файл g_ip.

12. Получить рекурсивно список файлов домашней директории, в имени которых есть буква «а», отсортированный без повторов.
    Результат занести в файл spisok_a.

Контрольные вопросы

1. Стандартные каналы ввода- вывода.
2. Операторы перенаправления каналов ввода- вывода.

Требование к отчету по выполненной работе

Отчет по работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель работы
3. Протокол использованных команд и результатов их работы (скриншоты) при выполнении пунктов задания работы. (Протокол оформить в формате .doc документа с сохранением нумерации пунктов заданий выполняемой работы).

Лабораторная работа N 5
Управление командной оболочкой
Создание простейшего скрипта

<u>Цель работы:</u> получить базовые знания языка shell. Научиться писать скрипты.

Теоретическое введение

Командный язык shell (в переводе - раковина, скорлупа, оболочка) оболочки фактически есть язык программирования очень высокого уровня. На этом языке пользователь LINUX осуществляет управление компьютером. Признаком того, что оболочка (shell) готова к приему команд, служит выдаваемая ею на экран строка приглашение. В простейшем случае это знак доллара "$".

Внимание: shell - это одна из многих команд LINUX, которая выполняется автоматически при входе в систему. Все командные оболочки находятся в каталоге */bin/* и имею имена в зависимости от их типа. Так, например, команда запуска оболочки Bourne Again Shell будет выглядеть так /bi*n/bash* (вспомните содержимое последнего поля файла /etc/passwd из темы №1). Первый "shell" вызывается автоматически при вашем входе. После этого вы можете вызывать на выполнение любые команды, в том числе и снова сам "shell[1]", который вам создаст новую оболочку внутри прежней.

Вывод строки приглашения и мерцание курсора на экране означает, что система готова к приему команд. Изучению основных команд управления данными (файлами и каталогами) была посвящена лабораторная работа №1 нашего лабораторного практикума.

Кроме команд управления каталогами и файлами существует целый ряд других команд (условная, цикла, выбора, ввода, вывода и т.д.), которые вместе и составляют язык программирования shell (или язык командной оболочки). Изучению этих команд и правилам их использования будут посвящены эта и следующая темы нашего практикума. Команды языка shell можно набирать в режиме интерактива (как это делалось в лабораторной работе №1), а можно помещать в специальный файл, который называется скриптом.

Скрипт - текстовый файл, содержащий команды оболочки. В первом случае (в режиме интерактива) командная оболочка создаст процесс и загружает туда названный файл. Во втором случае командная оболочка интерпретирует и выполняет команды из скрипта.

5.1 Переменные окружения

При запуске командной оболочки, как и любого другого процесса в системе, создастся *среда окружения,* которая представляет собой набор переменных, описывающих текущий сеанс работы с операционной системой. Список всех установленных переменных окружения можно получить, используя команду **env.**

[student@wpl student]$env<Enter>

Все переменные среды окружения доступны всем процессам пользователя, начиная с текущего. Некоторые переменные устанавливаются и используются самой командной оболочкой. Наиболее часто используемыми переменными являются:

Таблица 1- Переменные среды окружения.

Имя	**Значение**
UID	Содержит числовой идентификатор текущего пользователя. Инируется при запуске оболочки.
USER	Символьное имя пользователя
BASH	Имя используемой оболочки
HOME	Домашний каталог текущего пользователя.
PATH	Путь вызова. Список каталогов, разделенных двоеточием, в которых андная оболочка выполняет поиск команд, в случае если не задан путь.
PS1	Формат строки-приглашения (первая строка)
PS2	Формат строки-приглашения (вторая и последующие строки)
PWD	Текущий каталог

TERM	Тип используемого терминала
HOSTNAME	Сетевое имя компьютера
SECONDS	При каждом обращении к данной переменной возвращается ло секунд, прошедших с момента запуска оболочки. Если еменной было присвоено какое-либо значение, то при последующих

	ащениях возвращается число секунд с момента последнего своения плюс это число.

Если необходимо вывести значение какой-либо одной переменной окружения, то это можно сделать, используя команду echo, добавив при этом перед именем переменной символ "$".

Например, команда

[student@wpl student]$**echo $PATH<Enter>**

выведет на экран значение переменной окружения с именем **PATH.**

Внимание: Обратите внимание, что если вы напишите echo FRUIT, то на экран будет выведено слово FRUIT, а не значение переменной.

Установка новых и изменение значения существующих переменных окружения осуществляется путем *экспортирования:*

[student@wpl student]$ **export** ***имя переменной = значение.***

5.2 Формат командной строки

За формат командной строки (то, что будет выведено на экран в качестве приглашения к вводу команд) отвечают две *переменные окружения:* PS1 и PS2. Первая переменная PS1 настраивает основную первую командную строку, вторая - командную строку-продолжение (если перед нажатием <ENTER>в первой строке ввод команды не был завершен и в конце строки был введен символ '\'). В строку PS1 могут входить любые допустимые символы, команды терминала и специальные символьные последовательности (табл. 2).

Таблица 2- Специальные символы.

Спец.символ	**Выводимое значение**
\t	время в формате часы: минуты: секунды
\d	Дата в формате день недели месяц число-
\n	перевод строки.
\s	имя оболочки, базовое имя $0 (участок, следующий за нечным /)
\w	текущий рабочий каталог
\.W	базовое имя $PWD
\u	имя пользователя, под которым вы зарегистрированы

\h	hostname
\#	номер этой команды
\!	номер истории этой команды
\nnn	символ, соответствующий восьмиричному числу nnn
\$	если uid=0, то #, иначе $.
\\	обратная косая черта (backslash)
\[	начало последовательности невыводимых символов. жет использоваться для осуществления управления рминалом в приглашении.
\]	конец последовательности не выводимых символов.

Таким образом, для стандартного формата командной строки

[student@wpl student]$

будет справедливо утверждение – PS1= "[\u@\w \W]\$".

Если вы хотите, чтобы текущий пользователь был выделен зеленым цветом , надо использовать команду терминала. В таком случае, содержимое переменной PS1 будет иметь вид - **"[\[\033[34m\]\u\[\033[0m\]@\w \W]\$"**

5.3 Переменные оболочки.

Программируя на языке shell, пользователь имеет право создавать собственные переменные или так называемые переменные оболочки, объявлять которые предварительно не надо. Использование имени, отличного от имен системных переменных или переменных окружения уже интерпретируется как появление локальной переменной пользователя. Значение переменной присваивается следующем образом:

переменная=значение

Отличие такого присвоения от экспортирования состоит в том, что в данном случае переменная доступна только текущей программе и не может быть использована другими.

Доступ к значению переменной осуществляется также с помощью символа "$".

Например, команда:

[student@wpl student]$**FRUIT=apple <Enter>**

приведет к созданию переменной с именем FRUIT, а команда

[student@wpl student]$**echo $FRUIT <Enter>**

выведет на экран слово apple, записанное в эту переменную предыдущей командой. Переменная может принимать как текстовые (символьные), так и числовые значения.

Например, команда:

[student@wpl student]$**FRUIT=73 <Enter>**

приведет к записи в переменную FRUIT числа 73.

Внимание: Обратите внимание, что если вы напишите **echo FRUIT**, то на экран будет выведено слово FRUIT, а не значение переменной.

Чтобы удалить переменную используется команда **unset:**

[student@wpl student]$**unset имя <Enter>**

5.4 Подстановка команд, переменных и арифметических выражений

Под подстановкой понимается выполнение оболочкой определенного набора операций и интерпретация выходных данных как значения переменной, либо как параметра другой команды.

Подстановка команды осуществляется заключением её в обратные апострофы (на клавиатуре это клавиша со знаком ~),

например,

[student@wpl student]$**DATE=`date`**

В данном примере результат работы команды date (т.е. текущая системная дата) будут присвоены переменной с именем DATE.

Другой пример:

[student@wpl student]$ ***grep `id –un`* /etc/passwd**

В данном примере результат работы команды *id -un* (определение символьного имени текущего пользователя) будут использоваться как параметры команды *grep*.

Подстановка арифметического выражения осуществляется с помощью следующей конструкции: **$((выражение)).**

Например, строка:

[student@wpl student]$ **foo = $((((5+3*2) - 3)/2))**

присвоит переменной foo значение равное 4.

Подстановка переменных является ещё одним механизмом управления значениями переменных. Способы подстановки переменных перечислены в таблице 3.

Таблица 3- Команды подстановки

Форма подстановки	Описание
${ переменная:-значение}	Если переменная не определена или равна NULL, вместо него будет использовано значение. Реальное содержимое переменной при этом не изменяется.
${переменная:=значение}	Если переменная не определена или равна NULL, то ей присваивается значение.
${переменная:?сообщение}	Если переменная не определена или равна NULL, в стандартный поток ошибок (на экран монитора) выводится сообщение.
$ {переменная:+значение}	Если переменная установлена, вместо него используется значение. Реальное содержимое переменной не изменяется.

Примеры использования подстановок переменных –

[student@wpl student]$**echo ${DEBUG:+"работа в режиме отладки"}**

на экран выдается сообщение «работа в режиме отладки», если не определена переменная с именем DEBUG, в противном случае на экране ничего не появится.

5.5 Вывод и получение информации от пользователя

5.5. 1 Команда *printf*

В сценариях часто возникает необходимость вывести данные от пользователя на экран монитора. Для этого кроме команды **echo** можно использовать команда **printf.**

Команда **printf** записывается следующим образом:

[student@wpl student]$**printf “текст”**

или

[student@wpl student]$**printf $имя переменной**

5.5.2 Команда *read*

В сценариях часто возникает необходимость ввести данные от пользователя. Для этого используется команда read.

Команда **read** записывается следующим образом**:**

[student@wpl student]$ **read имя переменной <Enter>**

Данная команда читает строку, вводимую пользователем, и присваивает эту строку переменной. Ввод завершается нажатием клавиши <ENTER>. Ниже приведен пример использования команд **read и printf*:***

[student@wpl student]$ **printf "Завершить сеанс работы [введите Y или N]?"**
[student@wpl student]$ **read YN**
[student@wpl student]$ **echo "Вы ввели - $YN"**
[student@wpl student]$ **${YN:=yes}**
[student@wpl student]$ **echo $YN**

Протестируйте цепочку этих команд, с разными значениями для переменной YN, в том числе и не введя никакого значения.

5.6 Создание скрипта

Скрипт - текстовый файл, содержащий команды оболочки. Все изученные нами команды и команды, которые нам еще предстоит изучить в последующих темах, могут быть записаны в такого рода файл.

При сохранении скрипта можно к его имени добавлять расширение .sh.

Поскольку скрипт содержит команды, интерпретируемые и выполняемые командной оболочкой, ему необходимо дать статус исполняемого файла. Этот статус устанавливается путем изменения прав доступа к нему той категории пользователей, которой будет разрешено этот файл запускать на выполнение. Т.е. необходимо добавить такой категории пользователей право исполнять этот файл, использую команду **chmod,** изученную в лабораторной работе №2.

Готовый файл может быть запущен на исполнение следующим образом:

[student@wpl student]$ **./ имя оболочки имя_файла-скрипта**

Или возможно просто

[student@wpl student]$ **./имя_файла-скрипта**

Задания к лабораторной работе №5

Общее задание для всех

1. Определить тип используемой вами командной оболочки.

2. Вывести на экран значение каждой переменной среды окружения, описанной в таблице 1 этого теоретического раздела.

3. Используя переменную окружения HOME выполнить следующие действия списком:
   - перейти в домашний каталог,
   - выдать содержимое файла f3 , созданного в лабораторной работе №4

4. Используя команды printf и read, вывести приглашение пользователю «ввести команду». Ввести команду, записав ее в переменную пользователя.

5. Используя соответствующий оператор подстановки, выполнить проверку: если пользователь нажал <ENTER> без ввода команды, сообщить ему об ошибке.
   Выполнить ту команду, что он ввел.

6. Оформить предыдущие пункты 1-5 как скрипт и выполнить его.

Индивидуальные задания

Вариант 1.

1. Написать скрипт:

а) в текущем каталоге создать два подкаталога. Назвать их по имени текущего пользователя и терминала (использовать соответствующие переменные окружения);

б) ввести с клавиатуры два числа. При этом проверить, что переменные не пустые, в этом случае присвоить им числа 20 и 30. Подсчитать их сумму с помощью подстановки арифметических выражений;

в) создать файл в текущем каталоге. Результат сложения использовать в качестве имени файла;

г) сделать копию файла во втором подкаталоге и запретить группе право на чтение скопированного файла (применить числовой способ, причем остальные права должны сохраниться);

2. Написать второй сценарий, удаляющий следы первого.

Вариант 2.

1. Написать скрипт:

а) создать файл и подкаталог в текущем каталоге. Файл назвать по имени хоста (использовать переменную окружения);

б) сохранить оглавление каталога /var/log. При этом проверить, что переменная не пустая, иначе вывести сообщение “Каталог не существует”. Выдать оглавление на экран;

в) скопировать файл **.bash_profile** в созданный подкаталог. Разрешить пользователям, не входящим в группу, читать его (применить числовой способ, причем остальные права должны сохраниться);

2. Написать второй сценарий, удаляющий следы первого.

Вариант 3.

1. Написать скрипт:

а) сохранить в переменной путь до домашнего каталога. При этом проверить, что переменная не пустая, иначе вывести сообщение “Путь указан неверно”;

б) с помощью переменной создать два вложенных подкаталога;

в) запретить абсолютно всем входить во второй подкаталог (применить символьный способ записи, причем остальные права должны сохраниться);

г) скопировать файл со скриптом в один из подкаталогов. Вывести структуру на экран;

2. Написать второй сценарий, удаляющий следы первого.

Вариант 4.

1. Написать скрипт:

а) ввести с клавиатуры три строки: ваши фамилию, имя и отчество. При этом проверить, что переменные не пустые, в противном случае записать в них “Иванов”, “Иван”, ”Иванович”;

б) в домашнем каталоге создать подкаталог (назвать по фамилии), а в нем два файла (назвать по имени и отчеству);

в) разрешить всем остальным пользователям записывать в этот каталог, убрав право на чтение;

2. Написать второй сценарий, удаляющий следы первого.

Вариант 5.

1. Написать скрипт:

а) сохранить числовой идентификатор пользователя системы (использовать переменную окружения). Выполнить проверку того, что переменной присвоено значение, в противном случае вывести сообщение “Пишите правильно!”. Разделить это значение на число, введенное с клавиатуры (использовать арифметическую подстановку);

б) в домашнем каталоге создать подкаталог, назвав его в соответствии с полученным ответом;

в) переместить скрипт в этот подкаталог. Запретить абсолютно всем право записывать в этот файл (применить символьный способ, причем остальные права должны сохраниться).

2. Написать второй сценарий, удаляющий следы первого.

Вариант 6.

1. Написать скрипт:

а) вывести на экран сообщение “Введите команду:”. Результат записать в переменную, при этом проверить ее на пустоту, в этом случае присвоить ей оглавление домашнего каталога;

б) вывести на экран сообщение “Введите имя файла:”. Таким же образом выполнить проверку переменной, иначе присвоить ей какое-нибудь значение;

в) вывести на экран сообщение “Введите команду для создания файла:”;

г) выполните в терминале сначала вторую команду, затем первую.

2. Написать второй сценарий, в котором удалить созданный файл.

Вариант 7.

1. Написать скрипт:

а) в домашнем каталоге создать три вложенных друг в друга подкаталога, имена которых должны вводиться с клавиатуры. Выполнить проверку того, что переменным присвоены значения, иначе записать в них dir1, dir11, dir111 соответственно;

б) назначить одному из них такие права, чтобы полный доступ был только у владельца, а группу и остальных лишить всех прав (применить числовой способ записи);

в) в домашнем каталоге создать файл и скопировать его во все три подкаталога.

2. Написать второй сценарий, удаляющий следы первого.

Вариант 8.

Написать скрипт, в который включить команды для:

а) создания текстового файла. В нем прописать инструкции для вычисления следующих выражений согласно правилам утилиты **bc** (с точностью до тысячных):

1) перевести 524 из десятичной в двоичную систему счисления;
2) перевести 11010110001 из двоичной в десятичную систему;
3) перевести 65535 из десятичной в шестнадцатеричную систему;
4) найти значение выражения, переменные **x** и **y** вводятся с клавиатуры:

$$\frac{\sqrt{e^{xy} + \ln x^y}}{cos^2 x}$$

б) запуска соответствующей утилиты.

Вариант 9.

1. Написать скрипт, который включает команды для:

а) создания в домашнем каталоге еще одного файла-скрипта. Название файла вводится с клавиатуры, при этом выполнить проверку на наличие пустой переменной, в этом случае присвоить ей любое значение. В этом скрипте сохранить название используемого терминала, настройки языка, вывести эти значения на экран;

б) перемещения скрипта на Рабочий стол.

2.Написать второй сценарий, удаляющий следы первого.

Вариант 10.

1. Написать скрипт:

а) в домашнем каталоге создать подкаталог, а в нем файл. Их имена должны вводиться с клавиатуры и начинаться с точки. Выполнить проверку того, что переменным присвоены значения, в противном случае записать в них .dir, .file соответственно;

б) в этом файле набрать абзац текста, выдать на экран оглавление созданного подкаталога, затем содержимое файла;

в) отобрать у владельца право на чтение данного файла (применить числовой способ записи, у группы и остальных права должны остаться прежними);

г) еще раз вывести оглавление подкаталога и текст из файла.

2. Написать второй сценарий, удаляющий следы первого.

Вариант 11.

Написать скрипт:

ввести название какой-либо пользовательской команды Linux. Затем любую фразу. В том случае, если ничего не введено, присвоить переменным значения **“printf”** и **“about”** соответственно. Из первого раздела справочного man-руководства по заданной команде вывести на экран строки, содержащие данную фразу. Для этого следует обрабатывать файлы из каталога, содержащего справочные man страницы.

Вариант 12.

Написать скрипт:

а) из переменной окружения PATH извлечь все имеющиеся пути к директориям. Использовать подстановку параметров и операции со строками;

б) проверить инициализацию новых переменных, в этом случае вывести оглавление первых двух каталогов, для остальных выдать только общие сведения (права, владелец, дата изменения и т.д.).

Вопросы для контроля

1. Что такое скрипт-файл?
2. Что такое среда окружения? Зачем она нужна?
3. Как задать значение переменной окружения и как вывести его на экран?
4. Переменная оболочки. Отличие от переменной окружения.

Требование к отчету по лабораторной работе

Отчет по лабораторной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель лабораторной работы
3. Протокол использованных команд и результатов их работы, проиллюстрированные скриншотами экранов, при выполнении пунктов 1- 5 задания лабораторной работы. (Протокол оформить в формате .doc документа с сохранением нумерации пунктов заданий лабораторной работы).
4. Полный протокол создания скрипта (пункт 6 задания).
5. Текст скрипта.
6. Ответы на контрольные вопросы.

Лабораторная работа N 6

Управляющие конструкции командной оболочки Обработка параметров командной строки, передаваемых в скрипт (Способ 1)

Цель работы: Изучить управляющие (условные, циклические) конструкций управления командной оболочкой. Изучить способы взаимодействия командной оболочки и скриптов.

Теоретический материал

Командный язык shell (в переводе - раковина, скорлупа, оболочка) фактически есть язык программирования очень высокого уровня. На этом языке пользователь LINUX осуществляет управление компьютером. Следовательно, в этом языке обязательно должны быть управляющие операторы, позволяющие организовывать ветвление, циклические блоки алгоритмов управления.

6.1 Управляющие операторы

Командная оболочка bash позволяет организовать ветвление программы в зависимости от различных условий.

1. Условный оператор if-fi

Выражение if записывается следующем образом:

```
if список1 ; then
список2
elif список3 ; then
список4
else
список5
fi
```

Наличие операторов elif и else необязательно. В блоке *if-fi* может содержаться несколько elif, но только один оператор if-fi.

В приведенном выше выражении if сначала вычисляется список1. Если код завершения списка1 равен 0, что интерпретируется как истинность данного условия, вычисляется список2 и выражение заканчивается. В противном случае выполняется список3 и проверяется код его завершена. Если список3 возвращает

значение равное 0 (истина), выполняется список4 и выражение завершается. Если список3 возвращает ненулевое значение, выполняется список5.

Часто в блоке if-fi используют одну или несколько команд test, которая записывается следующим образом: *test выражение* или *[выражение]*. После оценки выражения команда test возвращает значение 0, либо 1. Опции test следующие:

Таблица 1- Опции команды test

Выражение	Значение
-d файл	Возвращает значение true, если указанный файл существует и является каталогом.
-e файл	Возвращает значение true, если указанный файл существует
-f файл	Возвращает true, если указанный файл существует и представляет собой обычный файл.
-L файл	Возвращает true, если указанный файл существует и представляет собой символьную ссылку.
-r файл	Возвращает true, если указанный файл существует и разрешен для чтения.
-s файл	Возвращает true, если указанный файл существует и имеет нулевой размер.
-w файл	Возвращает true, если указанный файл существует и разрешен для записи.
-x файл	Возвращает true, если указанный файл существует и является исполняемым файл.
-0 файл	Возвращает true, если указанный файл существует и принадлежит данному пользователю.
файл1 -nt файл2	Возвращает true, если файл1 последний раз был модифицирован позже, чем файл2
-z строка	Возвращает true, если указанная строка имеет нулевую длину
-n строка	Возвращает true, если указанная строка имеет ненулевую длину
строка1 == строка2	Возвращает true, если указанные строки совпадают.
! выражение	Возвращает true если указанное выражение false
выражение1 -a	Логическое AND двух выражений

выражение2	
выражение1 -o выражение2	Логическое OR двух выражений
выражение1 -eq выражение2	Возвращает true если выражение1 равно выражению2
выражение1 -ne выраженне2	Возвращает true если выражение1 неравно выражению2
выражение1 -lt выраженне2	Возвращает true если выражение1 меньше выражения2
выражение1 -1е выраженне2	Возвращает true если выражение1 меньше либо равно выраже-нию2
выражение1 -gt выражение2	Возвращает true если выражение2 меньше выражения1
выражение1 -ge выражение2	Возвращает true если выражение2 меньше либо равно выражению1

Ниже приведены примеры использования команды test в составе выражения if-fi:

if [-d $HOME/bin] ; then PATH="$PATH:$HOME/bin" ; fi

Эта управляющая команда if с помощью команды test проверяет существование каталога bin , хранящегося в домашнем каталоге пользователя. Имя домашнего каталога выбирается из переменной окружения HOME. В случае положительного ответа команды test в переменную PATH будет записана строка, содержимое которой приведено в кавычках.

if [-z "$DTHOME"] && [--d /home/student/dt] ; then SDTHOME=/home/student/dt ; fi

if [-z "$DTHOME" -a -d /home/student/dt] ; then SDTHOME=/home/student/dt ; fi

Попытайтесь разобрать смысл двух остальных команд if – fi самостоятельно.

2 Блок case-esac

Блок case-esac аналогичен оператору переключения switch языка Си и записывается следующим образом:

```
case слово   in
 шаблон 1)
          список1 команд
;;
шаблон_2)
         список2 команд
;;
?)  список3 команд
;;
esac
```

В данном случае *слово* - это строка символов или переменная, сравниваемая с шаблоном до тех пор, пока не будет выполнен критерий сравнения. Список команд, следующий за шаблоном, которому удовлетворяет *слово*, запускается на выполнение. За списком следует команда ;;, которая передает управление за пределы блока *case-esac*. Эта команда выполняет действия такие же, как и команда *break* в языке Си.

Если *слово* не удовлетворяет ни одному из шаблонов, выражение *case* завершается. Если необходимо выполнить какие-то действия по умолчанию,

следует включить в выражение шаблон "?", которому удовлетворяет любое значение *слова*.

В выражении *case-esac* должен присутствовать по крайней мере один шаблон. Максимальное число шаблонов неограниченно, В шаблоне могут использоваться символы, которые применяются в регулярных выражениях, в том числе оператор дизъюнкции "||" и &&- конъюнкции. Ниже приведен пример использования блока case-esac:

```
case  "$TER"  in
start)
       TERM = xtem
        echo “определена переменная TERM”
        echo  “ее значение =”
        echo  $TERM
       ;;
Out)
       DR=”$3”
       echo “определена переменная DR”
       echo “ее значение взято из спец. переменной $3”
       ;;
esac
```

В данном примере переменная TER может принять значения start или Out. Согласно ее значению в блоке case-esac будет выполняться либо блок с меткой start) или Out) соответственно. Любое другое значение переменной TER обработано не будет.

3 Циклические конструкции

Циклические конструкции применяются в том случае, когда надо многократно повторить одни и те же действия над разными данными.

Цикл for

Данный цикл записывается следующим образом:

```
for имя in список1 ;
do
список2
done
```

В цикле for переменной с указанным именем последовательно присваиваются все значения из списка1 и для каждого из этих значений выполняется список2.

Пример:

```
for i in 1 2 3 4 5 6 7 8 9 10 ;
 do
echo $i ;
 done
```

В данном фрагменте поочередно, для каждого значения переменной i от единицы до десяти будет срабатывать команда echo, выводящая значение i на экран монитора.

Цикл while

Данный цикл записывается следующим образом:

```
while список1
 do
список2
done
```

На каждой итерации этого цикла вычисляется список1 и до тех пор, пока он возвращает значение true, выполняется спискок2. Указывая в качестве списка1 /bin/true или ":" можно организовать бесконечный цикл.

Простой пример цикла while:

```
x=  1
while   [ $x -lt  10 ]
 do
echo $x
x=$((  $x+1  ))
 done
```

До тех пор, пока значение переменной x будет оставаться строго меньше 10 команда echo будет выводит эти значения на экран, а команда подстановки будет увеличивать значение x на единицу.

Цикл until

Данный цикл записывается следующим образом:

```
until  список1
 do
список2
done
```

На каждой итерации этого цикла вычисляется список1 и до тех пор, пока он возвращает значение false выполняется список2.

Простой пример цикла until;

```
x =  1
until   [ $x -ge  10 ]
do
echo  $x
x=$((  $x+1  ))
 done
```

6.2 Анализ параметров, передаваемых скрипту

Большинство программ в системе LINUX запускаются на исполнение следующим образом:

[student@wplstudent]$./***имяпрограммы*****[опции] [файлы/параметры]<Enter>**

Использование квадратных скобок говорит о том, что данная информация не является обязательной и может отсутствовать. Но, если опции и(или) параметры все же указаны, то их надо уметь принять и использовать в сценарии скрипта.

Обработку опций и параметров можно реализовать двумя способами:

1) реализовать проверку каждого из указанных параметров, используя специальные переменные

2) использовать команду **getopts.**

Способ 1. Специальные переменные

В оболочке bash определено несколько специальных переменных, часто используемых в скриптах:

Таблица 2- Специальные переменные

Имя	Назначение
$0(ноль)	Имя выполняемой команды. Для скрипта - это путь, указанный при его вызове.
$n	Переменные, соответствующие параметрам, заданным при вызове скрипта. Здесь n - десятичное число, соответствующее номеру параметра, (Первый параметр $1, второй $2 и т.д.)
$#	Число параметров, указанных при вызове скрипта.
$*	Строка параметров, заключенная в двойные кавычки.
$@	Все параметры, каждый заключен в двойные кавычки
$?	Статус завершения последней выполненной команды
$$	Номер процесса, соответствующего текущей оболочке.
$!	Номер процесса, соответствующий команде, запущенной в фоновом режиме.

Предположим, что у нас создан согласно всем правилам скрипт TEC.sh. При запуске этого скрипта на выполнение в него можно передать через список параметров любую информацию. Это могут быть и числа, и слова, и имена файлов и каталогов. Например, команда запуска нашего скрипта может иметь следующий вид:

[student@wpl student]$./**TEC.sh 5 обучение идет успешно CAT<Enter>**

Это будет соответствовать передаче в скрипт шести параметров, включая имя запускаемого скрипта. Все параметры будут записаны в спецпеременные (см. таблицу2). В спецпеременную с именем $0 будет помещен путь ./TEC.sh. В спецпеременную с именем $1 будет записано число 5. В спецпеременную с именем $2 будет помещено слово *обучение и т.д.* В спецпеременную с именем $5 будет записано слово CAT. В спецпеременную $# будет записано число 6, равное количеству параметров, обнаруженных в строке запуска скрипта. Смысл передачи и использования этих параметров внутри скрипта определяет сам автор.

Таблица 3 -Варианты заданий к лабораторной работе № 6

В №	**Задание**
1	**Н**аписать скрипт, в котором циклически реализовать меню в следующем виде: Выберите желаемый формат даты и времени: 1) ДД/ММ/ГГГГ; 2) ДД ММ, полное название дня недели; 3) ГГ со словом “год” номер дня недели со словом “день” номер недели в году со словом “неделя”; 4) часовой пояс по Гринвичу чч:мм; 5) количество дней, прошедших с начала года со словом “дней”, полное название месяца При выборе одного из пунктов на экране должна появиться дата в требуемом формате. Количество повторений цикла должно задаваться аргументом мандной строки и не должно превышать значение 5.
2	Написать скрипт, который принимает один аргумент - название каталога. Выполнить проверку наличия каталога на диске, проверку числа аргументов в строке, иначе выдать сообщение и завершить работу скрипта. Организовать цикл при помощи оператора **for,** в котором создать в другом каталоге символьные ссылки для каждого файла из аргумента-каталога. Имена ссылок должны оканчиваться на 01, 02, 03 и т.д. Для создания ссылок использовать команду **ln -s.**
3	Написать скрипт, аргументом которого является одна из команд (cat, less, nano, vi). Выполнить проверку:

	1) количества параметров в строке; 2) аргумент на пустоту. В операторе while интерактивно вводить названия файлов и выдавать экран их содержимое. Перед выдачей проверять наличие файлов на ске с правами пользователя на чтение, в случае отсутствия таковых дать сообщение. Предусмотреть выход из цикла. Вывод каждого файла отделять с помощью **********”.
4	Написать скрипт: Создать цикл, в котором на экран выводится меню: 1) Копирование файла 2) Перемещение файла 3) Создание файла При выборе соответствующего пункта запрашиваются полнительные параметры (имя файла и подкаталога) и выполняется ерация. В меню предусмотреть обработку недопустимой опции, а также ход из цикла. Для создания файла использовать команду touch. Количество повторений цикла должно задаваться аргументом мандной строки и не должно превышать значение 3.
5	Написать скрипт с тремя входными параметрами: **имя_файла имя_каталога число.** Проверить, чтобы файл имеет атрибут “исполняемый”. При помощи оператора **until** организовать цикл, в котором выполнить копирование **файла** в **каталог** определенное **число** раз. Каждую копию сопроводить сообщением “Создана копия 1”, “Создана копия 2” и т.д. Выполнить проверку количества параметров в строке, проверку наличия файла и каталога на диске, иначе выдать сообщение и завершить работу скрипта.
6	Написать скрипт с одним входным аргументом - названием каталога. В теле цикла просматривается состав каталога: для исполняемых файлов выводится однострочная справка (использовать утилиту **whatis**) с разделителем “**********” между строками, для остальных объектов выдается информация о типе файла (использовать утилиту **file**) с разделителем “----------” между строками. Сценарий должен проверять число аргументов в командной строке, если не указать аргумент, то использовать каталог **/usr/bin.**
7	Написать скрипт с одним входным аргументом - именем

	существующего файла. Если данный файл не существует или недоступен для чтения (объединить два условия), то выдать сообщение и выйти. С помощью цикла **until** происходит последовательная выдача на экран текста из этого файла (использовать команды **head** или **tail** с нужным параметром). Шаги пронумеровать и отделить символом “*”. .
8	Написать скрипт: Вывести на экран запрос: “Введите имя каталога”. Сообщение должно появляться постоянно до того момента, пока не будет введено название существующего каталога. С помощью цикла подсчитать: 1) количество символьных ссылок; 2) число файлов, владельцем которых является зарегистрированный пользователь; 3) количество подкаталогов; 4) количество блочных, символьных, сокетов и именованных файлов-каналов.
9	Написать скрипт, который: позволяет получить случайное число в определенном диапазоне. Скрипт принимает три аргумента: **нижнюю границу, верхнюю границу и количество чисел**. В цикле **until** на каждом шаге выдавать сообщение “На шаге **i** получено число **N**”. Генерация псевдослучайных чисел выполняется с помощью функции **$RANDOM.** В каждом запуске скрипта необходимо получать новые числа. Проверить число параметров в командной строке и их наличие.
10	Написать скрипт: При помощи оператора **for** организовать цикл для обработки файлов, размер которых лежит в указанном диапазоне. Скрипт принимает три аргумента:

	название каталога и два значения в байтах. Необходимо вычислить контрольную сумму (использовать команду **cksum**) для каждого файла, при этом на каждом шаге выводить сообщение “Контрольная сумма i-го файла равна N”. Проверить число параметров в командной строке и их наличие.
11	Написать скрипт, который: определяет общее количество исполняемых файлов (скриптов, утилит) в системе, используя список директорий из соответствующей переменной окружения. Подсчитать также число невыполняемых и файлов с установленным атрибутом SUID. Запустить скрипт от имени администратора и рядового пользователя. Сравнить результаты.
12	Написать скрипт, который: принимает переменное число аргументов (файлов). В случае отсутствия таковых выдать подсказку в виде *“Использование: название скрипта имя_файла1 …”*. Для каждого файла (подразумеваются текстовые) выдать информацию: 1) подсчитать длину каждого слова; 2) общий размер в печатаемых символах Разделителями слов считать пробелы.

Требование к отчету по лабораторной работе

Отчет по лабораторной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель лабораторной работы
3. Протоколы создания, тексты и результаты работы скриптов. (Протоколы оформить в формате .doc документов с подробными комментариями каждой использованной команды).
4. Ответы на контрольные вопросы.

Контрольные вопросы

1. Назначение условного оператора *if-fi*. Команда *test*
2 Назначение Блока *case-esac*
3. Назначение специальных переменных

Лабораторная работа N 7
Управляющие конструкции командной оболочки
Обработка параметров командной строки, передаваемых в скрипт
Способ 2

Цель работы: Изучить управляющие (условные, циклические) конструкций управления командной оболочкой. Изучить способы взаимодействия командной оболочки и скриптов.

Теоретический материал

Командный язык shell (в переводе - раковина, скорлупа, оболочка) фактически есть язык программирования очень высокого уровня. На этом языке пользователь LINUX осуществляет управление компьютером. Следовательно, в этом языке обязательно должны быть управляющие операторы, позволяющие организовывать ветвление, циклические блоки алгоритмов управления.

7.1 Управляющие операторы

Командная оболочка bash позволяет организовать ветвление программы в зависимости от различных условий.

1. **Условный оператор if-fi**

Выражение if записывается следующем образом:

if список1 ; then
список2
elif список3 ; then
список4
else
список5
fi

Наличие операторов elif и else необязательно. В блоке *if-fi* может содержаться несколько elif, но только один оператор if-fi.
В приведенном выше выражении **if** сначала вычисляется список1. Если код

завершения списка1 равен 0, что интерпретируется как истинность данного условия, вычисляется список2 и выражение заканчивается. В противном случае выполняется список3 и проверяется код его завершена. Если список3 возвращает значение равное 0 (истина), выполняется список4 и выражение завершается. Если список3 возвращает ненулевое значение, выполняется список5.

Часто в блоке **if-fi** используют одну или несколько команд test, которая записывается следующим образом: *test выражение* или *[выражение]*. После оценки выражения команда test возвращает значение 0, либо 1. Опции test следующие:

Таблица 1: Опции команды test

Выражение	Значение
-d файл	Возвращает значение true, если указанный файл существует и является каталогом.
-e файл	Возвращает значение true, если указанный файл существует
-f файл	Возвращает true, если указанный файл существует и представляет собой обычный файл.
-L файл	Возвращает true, если указанный файл существует и представляет собой символьную ссылку.
-r файл	Возвращает true, если указанный файл существует и разрешен для чтения.
-s файл	Возвращает true, если указанный файл существует и имеет нулевой размер.
-w файл	Возвращает true, если указанный файл существует и разрешен для записи.
-x файл	Возвращает true, если указанный файл существует и является исполняемым файл.
-0 файл	Возвращает true, если указанный файл существует и принадлежит данному пользователю.
файл1 -nt файл2	Возвращает true, если файл1 последний раз был модифицирован позже, чем файл2
-z строка	Возвращает true, если указанная строка имеет нулевую длину
-n строка	Возвращает true, если указанная строка имеет ненулевую длину

строка1 == строка2	Возвращает true, если указанные строки совпадают.
! выражение	Возвращает true если указанное выражение false
выражение1 -a выражение2	Логическое AND двух выражений
выражение1 -o выражение2	Логическое OR двух выражений
выражение1 -eq выражение2	Возвращает true если выражение1 равно выражению2
выражение1 -ne выраженне2	Возвращает true если выражение1 неравно выражению2
выражение1 -lt выраженне2	Возвращает true если выражение1 меньше выражения2
выражение1 -1е выраженне2	Возвращает true если выражение1 меньше либо равно выражению2
выражение1 -gt выражение2	Возвращает true если выражение2 меньше выражения1
выражение1 -ge выражение2	Возвращает true если выражение2 меньше либо равно выражению1

Ниже приведены примеры использования команды test в составе выражения if-fi:

if [-d $HOME/bin] ; then PATH="$PATH:$HOME/bin" ; fi

Эта управляющая команда if с помощью команды test проверяет существование каталога bin , хранящегося в домашнем каталоге пользователя. Имя домашнего каталога выбирается из переменной окружения HOME. В случае положительного ответа команды test в переменную PATH будет записана строка, содержимое которой приведено в кавычках.

2. Блок case-esac

Блок case-esac аналогичен оператору переключения switch языка Си и записывается следующим образом:

case *слово* in
шаблон 1)

```
список1 команд
;;
шаблон_2)
список2 команд
;;
?)  список3 команд
;;
esac
```

В данном случае *слово* - это строка символов или переменная, сравниваемая с шаблоном до тех пор, пока не будет выполнен критерий сравнения. Список команд, следующий за шаблоном, которому удовлетворяет *слово*, запускается на выполнение. За списком следует команда **;;**, которая передает управление за пределы блока *case-esac*. Эта команда выполняет действия такие же как и команда *break* в языке Си.

Если *слово* не удовлетворяет ни одному из шаблонов, выражение *case* завершается. Если необходимо выполнить какие-то действия по умолчанию, следует включить в выражение шаблон "?", которому удовлетворяет любое значение *слова*.

В выражении *case-esac* должен присутствовать по крайней мере один шаблон. Максимальное число шаблонов неограниченно, В шаблоне могут использоваться символы, которые применяются в регулярных выражениях, в том числе оператор дизъюнкции "||" и &&- конъюнкции. Ниже приведен пример использования блока case-esac:

```
case  "$TER"  in
start)
        TERM = xtem
         echo “определена переменная TERM”
         echo  “ее значение =”
         echo  $TERM
        ;;
Out)
        DR=”$3”
        echo “определена переменная DR”
        echo “ее значение взято из спец. переменной $3”
        ;;
esac
```

В данном примере переменная TER может принять значения start или Out. Согласно ее значению в блоке case-esac будет выполняться либо блок с меткой start) или Out) соответственно. Любое другое значение переменной TER обработано не будет.

3. Циклические конструкции

Циклические конструкции применяются в том случае, когда надо многократно повторить одни и те же действия над разными данными.

Цикл for

Данный цикл записывается следующим образом:

```
for  имя  in список1  ;
do
список2
done
```

В цикле **for** переменной с указанным именем последовательно присваиваются все значения из списка1 и для каждого из этих значений выполняется список2.

Пример:

```
for  i  in  1 2 3 4 5 6 7 8 9 10  ;
 do
echo  $i  ;
 done
```

В данном фрагменте поочередно, для каждого значения переменной i от единицы до десяти будет срабатывать команда **echo**, выводящая значение i на экран монитора.

Цикл while

Данный цикл записывается следующим образом:

```
while  список1
 do
список2
```

done

На каждой итерации этого цикла вычисляется список1 и до тех пор, пока он возвращает значение true, выполняется спискок2. Указывая в качестве списка1 /bin/true или ":" можно организовать бесконечный цикл.
Простой пример цикла while:

```
x=  1
while   [ $x -lt  10 ]
 do
echo $x
x=$((  $x+1  ))
 done
```

До тех пор, пока значение переменной x будет оставаться строго меньше 10 команда echo будет выводит эти значения на экран, а команда подстановки будет увеличивать значение x на единицу.

Цикл until

Данный цикл записывается следующим образом:

```
until  список1
 do
список2
done
```

На каждой итерации этого цикла вычисляется список1 и до тех пор, пока он возвращает значение false выполняется список2.
Простой пример цикла until;

```
x =  1
until   [ $x -ge  10 ]
do
echo  $x
x=$((  $x+1  ))
 done
```

7.2 Анализ параметров, передаваемых скрипту

Большинство программ в системе LINUX запускаются на исполнение следующим образом:

[student@wplstudent]$***имяпрограммы*** [опции] [файлы/параметры]<Enter>

Использование квадратных скобок говорит о том, что данная информация не является обязательной и может отсутствовать. Но, если опции и(или) параметры все же указаны, то их надо уметь принять и использовать в сценарии скрипта.

Обработку опций и параметров можно реализовать двумя способами:

1) реализовать проверку каждого из указанных параметров, используя специальные переменные

2) использовать команду getopts.

Способ 2. Команда getopts

Команда getopts вызывается внутри скрипта следующим образом:

getopts **строка_опций OPTION ;**

Опцией называется параметр командной строки, начинающийся с символов '-' или '--'. Причем, если используется один знак минуса, то опция считается односимвольной или короткой, если же два знака минуса, то - многосимвольной или длинной.

В строке опций указываются все возможные значения опции, которые должны быть учтены командой getopts.

Значение найденной опции присваивается **переменной OPTION ;**

Обработка опций командой **getopts** происходит следующим образом:

1. Выполняется проверка, не начинается ли параметр командной строки с символа '-'
2. Если символ '-' обнаружен, производится сравнение следующего за ‘-‘ символа с содержимым собственной строки опций команды **getopts**.
3. Если параметр совпадает с одним из символов в строке_опций, он записывается в **переменную OPTION;** . Если совпадение не обнаружено, переменной **OPTION** присваивается значение "?".

4. Если по сценарию вашего скрипта предусмотрена дальнейшая работа с этой переменной, то она будет выполнена.

5. Если вы предполагаете передавать в скрипт несколько опций, то следует команду **getopts** включить в циклическую команду (удобную для вас). Тогда описанная работа команды **getopts** повторяется для каждой введенной опции.

По окончании разбора опций, команда **getopts** возвращает значение, отличное от нуля, благодаря этому её удобно использовать в цикле. Кроме того, после окончания работы, команда **getopts** присваивает индекс последнего обработанного параметра системной переменной OPTIND.

Команда **getopts** позволяет также ввести и обработать опцию, для которой требуется дополнительный параметр, для этого надо в строке _опций команды **getopts** после соответствующего символа (предполагаемого значения опции) поставить знак **:.** В этом случае при разборе командной строки значение дополнительного параметра будет присвоено системной переменной OPTARG.

Рассмотрим фрагмент скрипта, в котором используется команда getopts.

```
VER=false
getopts vf:o: OPTION   ;

case  "$OPTION"   in
f)   INFILE="$OPTARG"
;;
o)   DUTFILE="$OPTARG"
;;
v)  VER=true
;;
\?)   echo   "ничего не найдено в строке”
 ;;
esac
```

Работу функции getopts в приведенном фрагменте следует понимать так:
В строке запуска скрипта функция getopts может обнаружить одну из трех опций **-f,-o** или **-v**. Если хотя бы один из этих символов будет обнаружен в строке запуска скрипта , найденное значение опции будет записываться в переменную OPTION. Оператор case будет искать среди своих разделов тот, который помечен таким же символом, как и записанная в OPTION опция. Если такой раздел будет обнаружен, помещенные в нем команды будут выполнены и блок case будет завершен. Если раздела с пометкой, аналогичной введенному значению опции, обнаружено не будет, case выберет группу операторов, помеченную символом ?). Кроме этого следует обратить внимание на то, что

опции **f** и **o** указаны как длинные (знак **:** после каждой из них в строке_опций), т.е. за каждым из них в строке запуска скрипта будет следовать их продолжение - дополнительный параметр, значение которого команда **getopts** будет сохранять в системной переменной OPTARG. Значением этой переменной можно пользовать в последующих командах скрипта.
Предположим, что приведенный выше фрагмент принадлежит скрипту с именем script.sh. Тогда, запустить на исполнение этот скрипт можно так:

[student@wpl student]**$./script.sh -o rtx.txt** <Enter>

или так

[student@wpl student]**$./script.sh -v** <Enter>

или так

[student@wpl student]**$./script.sh - -f CAT**<Enter>

или даже так

[student@wpl student]**$./script.sh -m**<Enter>

Но если вы хотите передавать в скрипт сразу несколько опций, например так

[student@wpl student]**$./script.sh -o rtx.txt -v -f CAT**<Enter>

Тогда приведенный выше фрагмент скрипта следовало бы изменить так:

```
VER=false
while getopts  vf:o: OPTION ;
do
case  "$OPTION"   in
f)   INFILE="$OPTARG"
;;
o)   DUTFILE="$OPTARG"
;;
v)  VER=true
;;
\?)   echo   "ничего не найдено в строке”
 ;;
esac
done
```

Первой будет обнаружена командой getopts опция ***-o.*** Эта опция помечена в команде getopts как длинная, т.е. **rtx.txt (** имя файла) – это дополнительный параметр опции ***–o***.

Имя файла rtx.txt будет записано в переменную OPTARG. Далее начнет работу блок case-esac и исполнит свой блок, помеченный меткой о). В этом блоке будет определена переменная DUTFILE, в которую запишется значение переменной OPTARG, т.е.имя файла rtx.txt.
Благодаря тому, что команда getopts включена в циклическую команду while, она продолжит разбор опций в командной строке и далее обнаружит опцию –v, значение которой запишет в переменную OPTION. А команда case-esac исполнит блок, помеченный меткой v) и т.д.

Варианты заданий к лабораторной работе № 7

№ варианта	Задание
1	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –m (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –d name создать каталог с именем name. г)на опцию –m выдать вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
2	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –f (с дополнительным параметром) и –m (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f name создать файл с именем name, если такого файла нет в домашнем каталоге. г)на опцию –m name выдать значение параметра name. д) предусмотреть обработку недопустимой опции
3	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу:

	в) на опцию –f name создать файл с именем name, даже если такой файла есть в домашнем каталоге. г)на опцию –d name создать каталог, если с таким именем нет. д) предусмотреть обработку недопустимой опции
4	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –f (короткая) и –m (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f выдать на экран приветственное сообщение. г)на опцию –m name проверить, есть ли файл с именем name. Если есть- выдать сообщение «есть», иначе «нет». д) предусмотреть обработку недопустимой опции
5	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f name выдать содержимое файла с именем name, если такой файла есть в домашнем каталоге. г)на опцию –d name создать каталог, если с таким именем нет. д) предусмотреть обработку недопустимой опции
6	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –f (короткая) и –m (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f выдать на экран приглашение «ввести команду», ввести и выполнить введенную команду. г)на опцию –m name проверить, если ли файл с именем name есть, выдать его содержимое на экран, иначе «нет». д) предусмотреть обработку недопустимой опции

7	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f name записать в файл с таким именем информацию, идущую с клавиатуры. г)на опцию –d name проверить, если ли каталог с именем name существует, переименовать его, присвоив ему ваше имя. д) предусмотреть обработку недопустимой опции
8	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –f (короткая) и –m (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f выдать на экран приглашение «ввести команду», ввести и выполнить введенную команду. г)на опцию –m name проверить, если ли файл с именем name есть, переименовать его, иначе «такого файла нет» д) предусмотреть обработку недопустимой опции
9	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f name вызвать редактор для внесения изменений , если такой файла есть в домашнем каталоге. г)на опцию –d name переименовать каталог своим именем, если такой каталог есть. д) предусмотреть обработку недопустимой опции
10	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу:

	Скрипту возможна передача опций –d (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f name вызвать на экран данные о вас: ФИО, номер группы. г)на опцию –d name вывести оглавление каталога, если такой каталог есть. д) предусмотреть обработку недопустимой опции
11	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –d (с дополнительным параметром) и –f (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f выдать на экран данные о вас: ФИО, номер группы. г)на опцию –d name вывести оглавление каталога, если такой каталог есть, иначе создать каталог с таким именем д) предусмотреть обработку недопустимой опции
12	Написать скрипт, анализирующий параметры командной строки с помощью команды getopts. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –f (короткая) и –m (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –f выдать на экран приглашение «ввести команду», ввести и выполнить введенную команду. г)на опцию –m name проверить, если ли файл с именем name есть, вывести его содержимое, иначе «такого файла нет» д) предусмотреть обработку недопустимой опции
13	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts.* Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –v (с дополнительным параметром) и –u (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –v name вывести содержимое файла с именем

	name, если этот файл принадлежит пользователю, ведущему сеанс работы. г)на опцию –u выдать числовой идентификатор пользователя, вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
14	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –u (с дополнительным параметром) и –k (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –u name запустить на выполнение файл с именем name, если этот файл есть и является исполнимым г)на опцию –k выдать сетевое имя вашего компьютера (использовать переменную окружения), вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
15	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –u (с дополнительным параметром) и –k (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –u name дозаписать информацию в файл с именем name, если этот файл есть и в него можно писать г)на опцию –k выдать сетевое имя вашего компьютера (использовать переменную окружения), вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
16	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –s (с дополнительным параметром) и –k (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –s param проверить, если значение param меньше 5, запросить ввод команды и выполнить ее, иначе выдать сообщение «ввод команды не возможен» г)на опцию –k вывести содержимое работающего скрипта, вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
17	Написать скрипт, анализирующий параметры командной строки

	с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –h (с дополнительным параметром) и –f (с дополнительным параметром). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –h name проверить совпадает ли имя текущего каталога сеанса с содержимым параметра name и если совпадает, вывести оглавление этого каталога с полной информацией о его содержимом. г)на опцию –f dat, проверить есть ли такой файл в текущем каталоге, выдать его содержимое, вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
18	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –u (с дополнительным параметром) и –k (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –u name проверить, является ли файл с именем name каталогом и если является, ввести новое имя и создать новый каталог с этим именем г)на опцию –k выдать символьное имя текущего пользователя, вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
19	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*. Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –u (с дополнительным параметром) и –k (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –u name проверить, является ли файл с именем name каталогом и если является, ввести новое имя и создать файл с этим именем г)на опцию –k выдать сетевое имя вашего компьютера (использовать переменную окружения), вашу Фамилию и имя д) предусмотреть обработку недопустимой опции
20	Написать скрипт, анализирующий параметры командной строки с помощью команды *getopts*.

	Скрипт должен выполнять следующую работу: Скрипту возможна передача опций –c (с дополнительным параметром) и –h (короткая). При обнаружении этих опций скрипт должен выполнить следующую работу: в) на опцию –c name проверить, является ли файл с именем name каталогом и если является, ввести два новых имени и создать новые каталоги с этими именами г)на опцию –h выдать символьное имя текущего пользователя, вашу Фамилию и имя д) предусмотреть обработку недопустимой опции

Требование к отчету по лабораторной работе

Отчет по лабораторной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель лабораторной работы
3. Протоколы создания, тексты и результаты работы скриптов. (Протоколы оформить в формате .doc документов с подробными комментариями каждой использованной команды).
4. Ответы на контрольные вопросы.

Контрольные вопросы

1. Назначение условного оператора ***if-fi.*** Команда ***test***
2 Назначение Блока ***case-esac***
3. Назначение функции ***getopts***

Лабораторная работа № 8
Понятие процесса, групп процессов, сеансов. Фоновое и интерактивное выполнение задач

Цель работы: получить основные навыки работы с процессами в системе.

8.1 Теоретическое введение

Для того чтобы программа могла быть запущена на выполнение, операционная система сначала должна создать окружение или среду выполнения задачи, куда относятся ресурсы памяти, возможность доступа к устройствам ввода/вывода и различным системным ресурсам, включая услуги ядра. Это окружение (среда выполнения задачи) получило название процесса. Мы можем представить процесс как совокупность данных ядра системы, необходимых для описания образа программы в памяти и управления ее выполнением. Мы можем также представить процесс как программу в стадии ее выполнения, поскольку все выполняющиеся программы представлены в UNIX в виде процессов. Процесс состоит из инструкций, выполняемых процессором, данных и информации о выполняемой задаче, такой как размещенная память, открытые файлы и статус процесса.

В то же время не следует отождествлять процесс с программой хотя бы потому, что программа может породить более одного процесса. Простейшие программы при выполнении представлены только одним процессом. Сложные задачи, например системные серверы (печати, FTP, Telnet), порождают в системе несколько одновременно выполняющихся процессов.

Операционная система LINUX является многозадачной. Это значит, что одновременно может выполняться несколько процессов, причем часть процессов могут являться образцами одной программы. Благодаря многозадачности операционной системы вы можете выполнять несколько задач не последовательно, одна за другой, а одновременно (Конечно сделать это можно только в том случае, если задачи не зависят друг от друга). Выполнение процесса заключается в точном следовании набору инструкций, который никогда не передают управление набору инструкций другого процесса. Процесс считывает и записывает информацию в раздел данных и в стек, но ему недоступны данные и стеки других процессов.

8.2 Типы процессов

Системные процессы

Системные процессы являются частью ядра операционной системы и всегда расположены в оперативной памяти. Они не имеют соответствующих им программ

в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы. Выполняемые инструкции и данные этих процессов находятся в ядре системы. Таким образом, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов. Системными процессами являются: **shed** (диспетчер свопинга), **vhand** (диспетчер страничного замещения), **bdfflush** (диспетчер буферного кэша) и **kmadaemon** (диспетчер памяти ядра) и т.д. К системным процессам следует отнести **init**, являющийся прародителем всех остальных процессов в LINUX. Хотя **init** не является частью ядра, и его запуск происходит из исполняемого файла (/etc/init), его работа жизненно важна для функционирования всей системы в целом.

Демоны

Демоны - это неинтерактивные процессы, которые запускаются обычным образом - путем загрузки в память соответствующих им программ (исполняемых файлов), и выполняются в *фоновом режиме.* Обычно демоны запускаются при инициализации системы и обеспечивают работу различных подсистем LINUX: системы терминального доступа, системы печати, системы сетевого доступа и сетевых услуг и т. п. Демоны не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть времени демоны ожидают пока тот или иной процесс запросит определенную услугу, например, доступ к файловому архиву или печать документа.

Прикладные процессы

К прикладным процессам относятся все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порожденные в рамках пользовательского сеанса работы. С такими процессами вы сталкивались чаще всего. Например, запуск команды ***ls*** породит соответствующий процесс этого типа (см. лабораторную N 1). Важнейшим пользовательским процессом является основной командный интерпретатор **(bash**), который обеспечивает вашу работу в LINUX. Он запускается сразу же после вашей регистрации в системе, а завершение работы **bash** приводит к отключению от системы.

Пользовательские процессы могут выполняться как в ***интерактивном,*** так и в ***фоновом режиме***, но в любом случае время их жизни (и выполнения) ограничено сеансом работы пользователя. При выходе из системы все пользовательские процессы будут уничтожены.

Интерактивные процессы монопольно владеют терминалом. Пока такой процесс не завершит свое выполнение, пользователь не сможет работать с другими приложениями. Вы сможете работать с другими приложениями, если в функции интерактивного процесса входит запуск на выполнение других программ. Примером такой задачи является командный интерпретатор **bash**, который считывает пользовательский ввод и запускает соответствующие задачи будучи сам

интерактивным процессом. Более типичным в данном контексте является процесс, порожденный, например командой ***ls***. Пока ***ls*** не завершит работу, вы не сможете вводить другие команды оболочки **bash**.

Фоновые процессы выполняются независимо от пользователя и время их выполнения совмещено со временем выполнения других фоновых и интерактивных процессов пользователя. Примером фоновых процессов могут служить процессы демоны.

Перевести выполнение задачи в фоновый режим можно одним из следующих способов:

1. Указать в конце командной строки символ амперсанд "&"
2. Во время выполнения программы нажать комбинацию клавиш <CTRL+Z>. Этим вы приостанавливаете выполнение текущей задачи в интерактивном режиме. Затем командой

b*g* *номер задачи*

можно запустить задачу на фоновое выполнение.

Любая фоновая задача будет приостановлена, если ей требуется доступ к консоли (например, для вывода или ввода информации).

Чтобы посмотреть список всех фоновых задач используется команда **jobs,** которая также выведет и текущее состояние задачи.

Чтобы перевести задачу, выполняющуюся в фоновом режиме, на передний план (в интерактивный режим), надо вызвать каманду **fg** и передать ей в качестве параметра номер фоновой задачи.

[student@wpl student]$**fg** *номер_задачи* <Enter>

Чтобы продолжить работу приостановленной задачи в фоновом режиме можно использовать команду

[student@wpl student]$**bg** *номер_задачи*<Enter>

8.3 Атрибуты процесса

Идентификатор процесса Process ID (PID)

Каждый процесс имеет уникальный идентификатор PID, позволяющий ядру системы различать процессы. Когда создастся новый процесс, ядро присваивает ему следующий свободный (т. е. не ассоциированный ни с каким процессом) идентификатор. Присвоение идентификаторов происходит по возрастающей, т. е. идентификатор нового процесса больше, чем идентификатор процесса, созданного перед ним. Если идентификатор достиг максимального значения, следующий процесс получит минимальный свободный PID и цикл повторяется. Когда процесс завершает свою работу, ядро освобождает занятый им идентификатор.

Идентификатор родительского процесса Parent Process ID (PPID)

Идентификатор процесса, породившего данный процесс.

Приоритет процесса (Nice Number)

Относительный приоритет процесса, учитываемый планировщиком при определении очередности запуска. Фактическое же распределение процессорных ресурсов определяется приоритетом выполнения, зависящим от нескольких факторов, в частности от заданного относительного приоритета. Относительный приоритет не изменяется системой на всем протяжении жизни процесса (хотя может быть изменен пользователем или администратором) в отличие от приоритета выполнения, динамически обновляемого ядром.

Терминальная линия (TTY)

Терминал или псевдотерминал, ассоциированный с процессом, если такой существует. Процессы-демоны не имеют ассоциированного терминала.

Реальный (RID) **и эффективный (EUID) идентификаторы пользователя**

Реальным идентификатором пользователя данного процесса является идентификатор пользователя, запустившего процесс. Эффективный идентификатор служит для определения прав доступа процесса к системным ресурсам (в первую очередь к ресурсам файловой системы). Обычно реальный и эффективный идентификаторы эквивалентны, т. е. процесс имеет в системе те же права, что и пользователь, запустивший его. Однако существует возможность задать процессу более широкие права, чем права пользователя путем установки флага SUID, когда эффективному идентификатору присваивается значение идентификатора владельца исполняемого файла (например, администратора).

Реальный (RGID) и эффективный (EGID) идентификаторы группы

Реальный идентификатор группы равен идентификатору первичной и текущей группы пользователя, запустившего процесс. Эффективный идентификатор служит для определения прав доступа к системным ресурсам и классу доступа группы. Так же как и для эффективного идентификатора пользователя, возможна его установка равным идентификатору групп владельца исполняемого файла (флаг SGID).

8.4 Группы процессов

Система LINUX позволяет легко объединять процессы в группы. Например, если в командной строке задано, что процессы связаны при помощи программного канала, они обычно помещаются в одну группу процессов. Примером группы может служить следующая команда:

[student@wpl student]**$ls -la | sort > ~/lists** <Enter>

Группы процессов удобны для работы с набором процессов в целом, с помощью механизма межпроцессного взаимодействия, который называется сигналами.

Каждая группа процессов обозначается идентификатором группы процессов (process group-id). Процесс, идентификатор которого совпадает с идентификатором группы процессов, считается *лидером* группы процессов.

8.5 Сеансы

В свою очередь, каждая группа процессов принадлежит к сеансу, В действительности сеанс относится к связи процесса с управляющим терминалом. Когда пользователь входит в систему, все процессы и группы процессов, которые он явно или неявно создает, будут принадлежать сеансу, связанному с их текущим терминалом. Сеанс обычно представляет собой набор из одной группы процессов переднего плана, использующей терминал, и одной или более групп фоновых процессов. Каждый сеанс также обозначается идентификатором, который совпадает с PID процесса лидера.

8.6 Состояние процесса

Работоспособный (runnable) процесс. Если процесс в текущий момент выполняет какие-либо действия или стоит в очереди на получение кванта времени на центральном процессоре, он называется работоспособным и обозначается символом R. Только работоспособные процессы потребляют процессорное время или конкурируют за этот ресурс. Их нельзя полностью выгрузить на диск, так как часть программного кода и сегмента данных должны находиться в памяти. Если суммарный размер резидентных частей работоспособных программ превышает объем оперативной памяти ЭВМ, ОС начинает интенсивно выполнять операции свопинга, что приводит к катастрофическому (на два-три порядка) падению быстродействия системы.

Ожидающий (спящий, sleeping) процесс. Это состояние обозначается символом S и возникает после того, как процесс инициирует системную операцию, окончания которой он должен дождаться. К таким операциям относятся ввод/ вывод, стечение заданного интервала времени, завершение дочернего процесса и т. д. Операции вывода на жесткие диски в UNIX достаточно эффективно буферизуются системой, поэтому даже интенсивно обменивающийся информацией с жестким диском процесс вряд ли будет находиться в состоянии ожидания. Операции вывода на медленные носители вроде гибких магнитных дисков и магнитных лент будут приводить к ожиданию программой завершения длительных операций. Системные процессы-демоны большую часть времени проводят в состоянии ожидания и не потребляют процессорное время. Процесс в состоянии ожидания можно полностью выгрузить на диск из оперативной памяти компьютера.

Остановленный (stopped) процесс. Процесс можно остановить в любое время и после останова продолжить его выполнение. Остановленный процесс не потребляет основные ресурсы компьютера - процессор, так как процесс не работает, и оперативную память, так как процесс можно целиком выгрузить на диск. Пользователь может остановить процесс, например, чтобы дать возможность другому процессу использовать больший объем оперативной памяти или быстрее завершиться. Обычный пользователь может остановить и продолжить только свой процесс, а суперпользователь - любой. Операционная система останавливает фоновые процессы в случае, когда они пытаются ввести/вывести данные с терминала. Программа-отладчик может останавливать отлаживаемый процесс в контрольных точках. Это состояние обозначается символом T,

Завершившийся (зомби, "zombie") процесс. После завершения процесса информация о нем должна быть удалена операционной системой из таблицы процессов. В ОС UNIX такая операция возможна только после того, как родительский процесс выполнит системную операцию ожидания завершения дочернего процесса. Это связано с используемой в ОС UNIX системой учета потребляемых ресурсов. Если родительский процесс выполняется параллельно с дочерним, ОС вынуждена хранить в таблице процессов запись о завершившемся процессе, хотя он реально уже не существует и не потребляет ресурсы ЭВМ. Завершившийся процесс обозначается символом Z.

8.7 Получение информации о процессах системы

Команда **ps** (process status) позволяет вывести список процессов, выполняющихся в системе, и их атрибуты (см. табл. 8).

Таблица 1- Опции команды **ps**

Опция	Действие
-a	Выдать все процессы системы, включая лидеров сеансов
-d	Выдать все процессы системы, исключая лидеров сеансов
-e	выдать все процессы системы
-X	выдать процессы системы, не имеющие контрольного терминала
-o	указать формат вывода. Формат задастся в виде символьной строки, поля которой разделяются символом ','. Информацию о формате см. в табл 2
-U	выдать процессы, принадлежащие указанному пользователю. Пользователь задается в символьном виде.

Таблица 2- Формат вывода команды ps

Форма	Значение
F	статус процесса (системный процесс, блокировки в памяти и т.д.
S	Состояние процесса (0 выполняется процессором, S - находится в состоянии сна, R -готов к выполнению, I - создается, Z - зомби)
UID	идентификатор (имя) пользователя
PID	идентификатор процесса
PPID	Идентификатор родительского процесса
PRI	текущий динамический приоритет процесса
NI	Значение nice number процесса
TTY	управляющий терминал процесса ('?' - означает отсутствии равляющего терминала)
TIME	суммарное время выполнения процесса процессором
STIME	Время создания процесса (может отличаться от времени запуска ограммы)
COMMAND	имя команды, соответствующей процессу
SESS	Сеанс процесса
PGRP	группа процесса

Например, список запущенных процессов вашей системы можно получить, используя команду

[student@wpl student]$ **ps -e** <Enter>

Информацию о группах и сессиях процессов можно получить так:

[student@wpl student]$ **ps -axo pid, pgrp, session, command** <Enter>

Можно также посмотреть "дерево" процессов, используя команду **pstree.**

Чтобы посмотреть использование ресурсов процессами в динамике можно использовать команду **top**. Выход из просмотра осуществляется нажатием клавиши <Q>.

Задание к лабораторной работе № 8

Общее задание для всех

1. Используя команду **ps** вывести информацию обо все процессах системы и ответить на следующие вопросы: сколько процессов в системе? сколько процессов принадлежит Вам?

2. Используя форматный вывод команды **ps** получить дерево процессов, принадлежащих вашему сеансу.

3. Используя команду **pstree** и перенаправление канала вывода вывести дерево процессов в файл ***~ /pstrees.*** Проанализировать результат работы команды. Сравнить с результатами предыдущего пункта.

4. Используя команду **top** описать наиболее активные процессы в системе.

5. Выполнить команду **man** в фоновом режиме.

6. Используя команду **jobs** посмотреть состояние фоновых задач.

7. Перевести задачу 1 в интерактивный режим и объяснить, почему выполнение указанной команды было автоматически приостановлено.

8. Ответить на вопрос: если запустить в фоновый режим следующий набор команд **ls | sort** будут ли указанные процессы ***(ls и sort***) входить в одну группу или в разные?

Таблица 3- Индивидуальные варианты задания

№	Задание
1	Написать сценарий-отладчик: 1) выполняющий трассировку значения переменной; организовать цикл, в котором последовательно печатать названия всех игналов (для этого использовать встроенную команду bash для отладки); 3) на выходе из сценария по соответствующему сигналу выдать сообщение.
2	Написать сценарий: 1) организовать цикл для выдачи списка выполняющихся процессов в расширенном формате; 2) выводить значения только из колонок PID, LWP, NLWP, TIME, CMD; Использовать операцию подстановки процессов.
3	Написать сценарий: 1) создать секундомер с выводом на экран изменений каждую кунду; 2) предусмотреть в скрипте аварийное завершение путем нажатия ответствующей комбинации клавиш; 3) на выходе выдать сообщение “Скрипт завершен на N секунде”.
4	Написать сценарий, который: 1) каждые N секунд в течение M секунд выводит на экран текущее время; 2) запускается в фоновом режиме. Написать второй сценарий, который ожидает завершения первого задания, после чего печатает сообщение “Продолжительность ожидания составила N секунд”.
5	Написать сценарий, который: 1) запускается в фоне; 2) отслеживает все процессы, запущенные в фоне после него и принудительно завершает их;

	3) вывести на экран PID завершенных процессов или номер задания.
6	Написать сценарий который запускает долгоиграющий процесс (например, поиск строки **namespace** в файлах .h) в фоне и выводит меню: 1) Остановить процесс; 2) Продолжить выполнение; 3) Перевести в фоновый режим; 4) Перевести в активный режим; 5) Завершить процесс; **6)** Завершить принудительно
7	Написать сценарий, который: 1) датчиком случайных чисел формирует сигнал и посылает его родительскому процессу сценария**;** 2) в случае отсутствия реакции на первый сигнал генерирует новый и т.д.
8	Написать сценарий, который: 1) определяет зависшие процессы в системе (условно полагать, что такие процессы длятся более 1 минуты); 2) пытается завершить их обычным образом; **3)** в отсутствии реакции на первый сигнал принудительно завершает процесс.
9	Написать сценарий: 1) который печатает сообщение “Ваш любимый фрукт?” 2) по сигналу таймера ожидает ответа в течение N секунд; 3) по истечении таймаута пишет “Ваше время истекло. Попробуйте еще раз”; 4) повторно выдавать сообщение, увеличив время ожидания в два раза.

	Число N передавать в виде аргумента командной строки.
10	Написать сценарий: 1) определяющий процессы, номера которых находятся в заданном диапазоне; 2) следующую информацию о каждом процессе записывать в отдельные файлы: название, состояние, количество потоков, номер процесса, идентификатор родительского процесса. Начальное и конечное значение PID передавать в виде аргументов командной строки.

Контрольные вопросы

1. Отличие процесса от программы или задачи?
2. Типы процессов
3. Группы процессов? Зачем используются?
4. Сеансы? Зачем используются?
5. Фоновое выполнение задач. Способы перевода задачи в фоновый режим.
6. Атрибуты процесса
7. Выделение идентификатора процессу?
8. Что такое приоритет процесса? Зачем он нужен?
9. Что такое идентификатор родительского процесса? Зачем он используется?
10. Что такое терминальная линия? Какие процессы не используют терминальных линий?
11. Состояние процесса

Требование к отчету по работе

Отчет по выполненной работе должен содержать следующие разделы:

1. Титульный лист
2. Название, цель работы
3. Протокол использованных команд и результатов их работы при выполнении пунктов задания лабораторной работы. (Протокол оформить в формате .doc документа с сохранением нумерации пунктов заданий работы).

СПИСОК ЛИТЕРАТУРЫ

1. Таненбаум Э. Операционные системы: разработка и реализация. 3-е изд. / Э. Таненбаум, А. Вудхалл. – СПб.: Питер, 2007. – 704 с. – ISBN 978-5-469-01403-4.
2. Стивенс У. UNIX: взаимодействие процессов / У. Стивенс. – СПб.: Питер, 2003. – 576 с. – ISBN 5-318-00534-9.
3. Иванов Н.Н. Программирование в Linux. Самоучитель. / Н.Н. Иванов. – СПб.: БХВ-Петербург, 2007. – 416 с. – ISBN 978-5-9775-0071-5.
4. Немет Э. UNIX: руководство системного администратора / Э. Немет, Г. Снайдер, С. Сибасс, Т.Р. Хейн. – Киев: BHV, 2002. – 928 с. – ISBN 966-552-106-3.
5. Робачевский А. Операционная система UNIX / А. Робачевский. – СПб: BHV, 2002. – 528 с. – ISBN 5-8206-0030-4.
6. Изучаем Linux : практикум / О. И. Моренкова ; Сибирский государственный университет телекоммуникаций и информатики. - Новосибирск : СибГУТИ, 2022. - 121 с. : ил. - Загл. с титул. экрана. - Электрон. версия без печ. публикации. - URL: http://ellib.sibsutis.ru/ellib/2020/896_Morenkova _O.I. _Izuchaem _.pdf. - Режим доступа: по паролю. - Библиогр.: с. 101. - : ~Б. ц. - Текст : электронный.

   Авт. договор № 761 от 19.02.2022 г.
7. Шоттс, У. Командная строка Linux. Полное руководство. 2-е межд. изд. — СПб.: Питер, 2020. — 544 с.: ил.
8. Тейлор Д., Перри Б. Сценарии командной оболочки. Linux, OS X и Unix. 2-е изд. — СПб.: Питер, 2017. — 448 с.: ил.

Список дополнительной литературы

1. Гончарук С.В. Администрирование ОС Linux : учебное пособие / Гончарук С.В.. — Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 163 с. — ISBN 978-5-4497-0299-9. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/89414.html (дата обращения: 30.06.2023). — Режим доступа: для авторизир. Пользователей

Информационное обеспечение (в т.ч. интернет- ресурсы).

1. Сафонов, В. О. Основы современных операционных систем : учебное пособие / В. О. Сафонов. — 3-е изд. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 826 с. — ISBN 978-5-4497-0552-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/94855.html (дата обращения: 08.07.2020). — Режим доступа: для авторизир. пользователей
2. Виртуальная энциклопедия "Linux по-русски". Режим доступа: http://rus-linux.net.

ПРИЛОЖЕНИЕ 1.
КОДЫ ОШИБОК

Код ошибки	Сообщение	Описание
E2BIG	Arg list too long	Размер списка аргументов, переданных системному вызову *exec*, плюс размер экспортируемых переменных окружения превышает ARGMAX байт.
EACCESS	Permission denied	Попытка доступа к файлу с недостаточными правами для данного класса (определяемого эффективными UID и GID процесса и соответствующими идентификаторами файла).
EAGAIN	Resource temporarily unavailable	Превышен предел использования некоторого ресурса, например, переполнена таблица процессов, или пользователь превысил ограничение по количеству процессов с одинаковым UID. Причиной также может являться недостаток памяти или превышение соответствующего ограничения.
EALREADY	Operation already in progress	Попытка операции с неблокируемым объектом, уже обслуживающим некоторую операцию.
EBADF	Bad file number	Попытка операции с файловым дескриптором, не адресующим никакой файл; также попытка операции чтения или записи с файловым дескриптором, полученным при открытии файла на запись или чтение, соответственно.
EBADFD	File descriptor in bad state	Файловый дескриптор не адресует открытый файл или попытка операции чтения с файловым дескриптором, полученным при открытии файла только на запись.
EBUSY	Device busy	Попытка монтирования устройства (файловой системы), которое уже примонтировано; попытка размонтировать файловую систему, имеющую открытые файлы; попытка обращения к недоступным ресурсам (семафоры, блокираторы и т.п.).
ECHILD	No child processes	Вызов функции *wait* процессом, не имеющим дочерних процессов или процессов, для которых уже был сделан вызов *wait*.
EDQUOT	Disk quota exceeded	Попытка записи в файл, создание каталога или файла при превышении квоты пользователя на дисковые блоки, попытка создания файла при превышении пользовательской квоты на число *inode*.
EEXIST	File exists	Имя существующего файла использовано в недопустимом контексте, например, сделана попытка создания символической ссылки с именем уже существующего файла.
EFAULT	Bad address	Аппаратная ошибка при попытке использования системой аргумента функции, например, в качестве указателя передан недопустимый адрес.
EFBIG	File too large	Размер файла превысил установленное ограничение RLIMIT_FSIZE или максимально допустимый размер для данной файловой системы.

EINPROGRESS	Operation now in progress	Попытка длительной операции (например, установление сетевого соединения) для неблокируемого объекта.
EINTR	Interrupted system call	Получение асинхронного сигнала, например, сигнала SIGINT или SIGQUIT, во время обработки системного вызова. Если выполнение процесса будет продолжено после обработки сигнала, прерванный системный вызов завершится с этой ошибкой.
EINVAL	Invalid argument	Передача неверного аргумента системному вызову. Например, размонтирование устройства (файловой системы), которое не было примонтировано. Другой пример – передача номера несуществующего сигнала системному вызову *kill*.
EIO	I/O error	Ошибка ввода-вывода физического устройства.
EISDIR	Is a directory	Попытка операции, недопустимой для каталога, например, запись в каталог с помощью вызова *write*.
ELOOP	Number of symbolic links encountered during path name traversal exceeds MAXSYMLINKS	При попытке трансляции имени файла было обнаружено недопустимо большое число символических ссылок, превышающее значение MAXSYMLINKS.
EMFILE	Too many open files	Число открытых файлов для процесса превысило максимальное значение OPENMAX.
ENAMETOOLONG	File name too long	Длина полного имени файла (включая путь) превысила максимальное значение PATHMAX.
ENFILE	File table overflow	Переполнение файловой таблицы.
ENODEV	No such device	Попытка недопустимой операции для устройства. Например, попытка чтения устройства только для записи или операция для несуществующего устройства.
ENOENT	No such file or directory	Файл с указанным именем не существует или отсутствует каталог, указанный в полном имени файла.
ENOEXEC	Exec format error	Попытка запуска на выполнение файла, который имеет права на выполнение, но не является файлом допустимого исполняемого формата.
ENOMEM	Not enough space	При попытке запуска программы (*exec*) или размещения памяти (*brk*) размер запрашиваемой памяти превысил максимально возможный в системе.
ENOMSG	No message of desired type	Попытка получения сообщения определённого типа, которого не существует в очереди.
ENOSPC	No space left on device	Попытка записи в файл или создания нового каталога при отсутствии свободного места на устройстве (в файловой системе).
ENOSR	Out of stream resources	Отсутствие очередей или головных модулей при попытке открытия устройства STREAMS. Это состояние является временным. После освобождения соответствующих ресурсов другими процессами операция может пройти успешно.
ENOSTR	Not a stream device	Попытка применения операции, определённой для

		устройств типа STREAMS (например, системного вызова *putmsg* или *gefmsg*), для устройства другого типа.
ENOTDIR	Not a directory	В операции, предусматривающей в качестве аргумента имя каталога, было указано имя файла другого типа (например, в пути для полного имени файла).
ENOTTY	Inappropriate ioctl for device	Попытка выполнения системного вызова *ioctl* для устройства, которое не является символьным.
EPERM	Not owner	Попытка модификации файла способом, разрешённым только владельцу и суперпользователю и запрещённым остальным пользователям. Попытка операции, разрешённой только суперпользователю.
EPIPE	Broken pipe	Попытка записи в канал (*pipe*), для которого не существует процесса, принимающего данные. В этой ситуации процессу обычно отправляется соответствующий сигнал. Ошибка возвращается при игнорировании сигнала.
EROFS	Read-only file system	Попытка модификации файла или каталога для устройства (файловой системы), примонтированного только для чтения.
ESRCH	No such process	Процесс с указанным PID не существует в системе.

Ольга Ильинична Моренкова
Алексей Юрьевич Голошубов

Операционные системы. Изучаем LINUX
Практикум

Редактор:
Корректор:

Подписано в печать _____ Формат бумаги 60 х 84/16, отпечатано на ризографе, шрифт № 10, изд. л. 6,3 заказ №_____, тираж – 100 экз., СибГУТИ. 630102, г. Новосибирск, ул. Кирова, д. 86