



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники (ВТ)

ЛАБОРАТОРНЫЕ РАБОТЫ ПО ДИСЦИПЛИНЕ

Архитектура ВМиС

(наименование дисциплины)

09.03.01 – «Информатика и

Направление подготовки

ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»

(код и наименование)

Профиль

«Вычислительные машины и системы»

(код и наименование)

Форма обучения

очная

(очная, очно-заочная, заочная)

Программа подготовки

академический

(академический, прикладной бакалавриат)

Квалификация выпускника

Бакалавр

Москва 2020

Содержание

1. Лабораторная работа.	3
2. Лабораторная работа.....	15
3. Лабораторная работа.....	19

Основные требования к выполнению работ

Все этапы выполнения работ необходимо фиксировать, создавая скриншоты на каждом этапе выполнения. Работы выполняются под уникальной пользовательской учетной записью. Соблюдать синтаксис и правильность написания команд, имен файлов и каталогов, а также содержание создаваемых файлов. Все работы выполняются исключительно в окне терминала. Отчет должен содержать цель работы, основную часть (скриншоты и комментарии о выполненных действиях), вывод.

Лабораторная работа №1

Виртуализация: установка и настройка операционной системы UbuntuLinux на виртуальной машине OracleVirtualBox

Введение

Виртуальная машина – это программа, которая эмулирует реальный (физический) компьютер со всеми его компонентами (жёсткий диск, DVD/CD привод, BIOS, сетевые адаптеры и т.д.). На такой виртуальный компьютер можно установить, например, операционную систему, драйверы, программы и т.д. Таким образом, вы можете запустить на своем реальном компьютере еще несколько виртуальных компьютеров, с такой же или другой операционной системой. Вы можете без проблем осуществить обмен данными между вашим реальным и виртуальным компьютером.

Зачем нужна виртуальная машина?

Виртуальную машину используют для различных целей и задач:

- установка второй/другой операционной системы;
- тестирование программного обеспечения;
- безопасный запуск подозрительных программ;
- эмуляция компьютерной сети;
- запуск приложений, которые нельзя запустить в вашей операционной системе.

Существует достаточно обширный список средств для создания и управления виртуальными машинами. В данной лабораторной работе мы рассмотрим одно из них – виртуальную машину VirtualBox фирмы Oracle.

VirtualBox – это бесплатный программный продукт, с помощью которого можно создавать виртуальные машины и установить на них все самые популярные операционные системы. VirtualBox поддерживает работу с Windows, Linux, FreeBSD, Mac OS.

Цель лабораторной работы

Целью данной лабораторной работы является получение практических навыков установки и создания виртуальных машин в Oracle VirtualBox, а также изучение принципов инсталляции и начальной настройки операционной системы Ubuntu Linux.

В результате выполнения лабораторной работы студенты познакомятся с процессом установки на персональный компьютер виртуальной машины OracleVirtualBox, получат представление о процессе создания и настройки виртуального окружения. На примере операционной системы UbuntuLinux будет выполнен процесс установки и базовой настройки операционной системы.

Задание

Перед выполнением лабораторной работы следует ознакомиться с архитектурой операционной системы Linux, представленной в приложении. По окончании необходимо ответить на контрольные вопросы по теме лабораторной работы.

Для выполнения лабораторной работы необходимо скачать с официального сайта компании Oracle дистрибутив виртуальной машины VirtualBox и выполнить установку ска-

чанного дистрибутива на компьютер. После установки необходимо с помощью инструментов, предоставляемых VirtualBox создать и настроить виртуальную машину, и установить на нее операционную систему UbuntuLinux. Дистрибутив для установки необходимо скачать из интернета. В процессе создания виртуальной машины необходимо определить расположение файлов виртуальной машины на компьютере, выделить объем оперативной памяти, видеопамяти, жесткого диска, необходимых для функционирования устанавливаемой операционной системы. Задать количество ядер центрального процессора, используемых виртуальной машиной и предельный уровень загрузки процессора. При установке операционной системы необходимо задать способ разбиения жесткого диска на логические разделы.

Примечание: Все этапы выполнения лабораторных работ необходимо фиксировать, делая снимки экрана терминала, с последующим внесение в отчет с комментариями.

Выполнение лабораторной работы

1. Скачать дистрибутив Oracle VirtualBox

Для выполнения шага необходим компьютер, имеющий доступ в интернет.

Запускаем браузер.

В адресной строке браузера вводим <https://www.virtualbox.org/wiki/Downloads>

На открывшейся странице переходим по ссылке Windows hosts.

Сохраняем установочный файл на жесткий диск.

(Альтернативно можно скачать, набрав в адресной строке браузера прямую ссылку на дистрибутив. На момент составления методички прямая ссылка выглядела следующим образом: <http://download.virtualbox.org/virtualbox/5.2.2/VirtualBox-5.2.2-119230-Win.exe>)



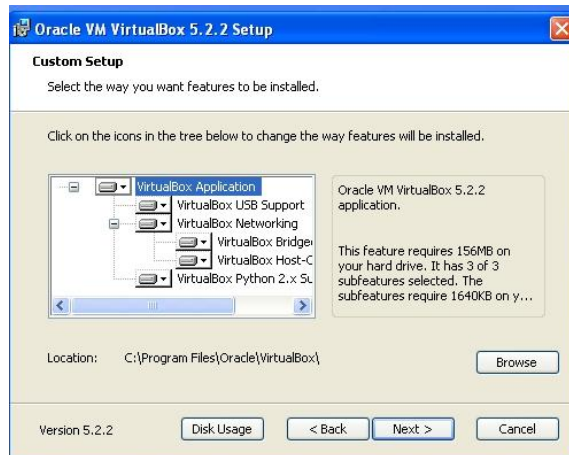
2. Установить Oracle VirtualBox на компьютер

Для установки OracleVirtualBox на компьютере необходимо запустить скачанный на первом шаге установочный файл дистрибутива.



В появившемся диалоге необходимо выбрать компоненты для установки. Для удобст-

ва на этапе обучения рекомендуется в диалоге выбрать установку всех предложенных компонентов.



По завершении установки на рабочем столе компьютера будет создан ярлык для запуска виртуальной машины OracleVirtualBox



3. Получить дистрибутив операционной системы Ubuntu Linux

Для выполнения шага необходим компьютер, имеющий доступ в интернет.

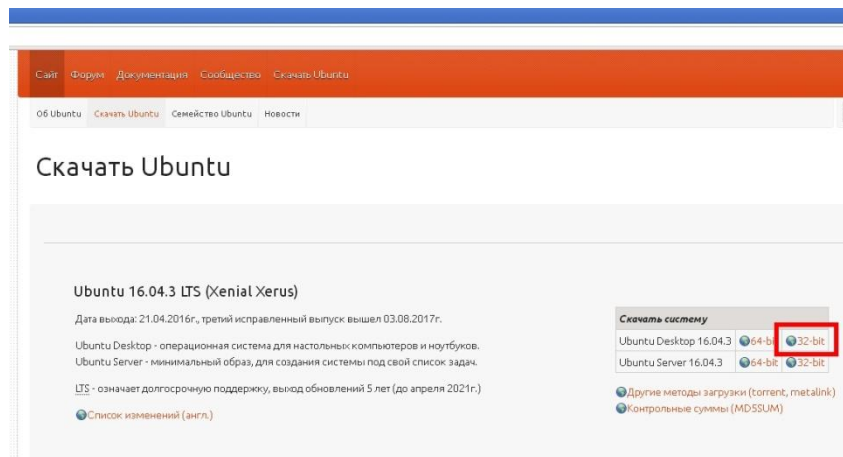
Запускаем браузер.

В адресной строке браузера вводим <http://www.ubuntu.ru/>

На открывшейся странице переходим по ссылке для скачки.



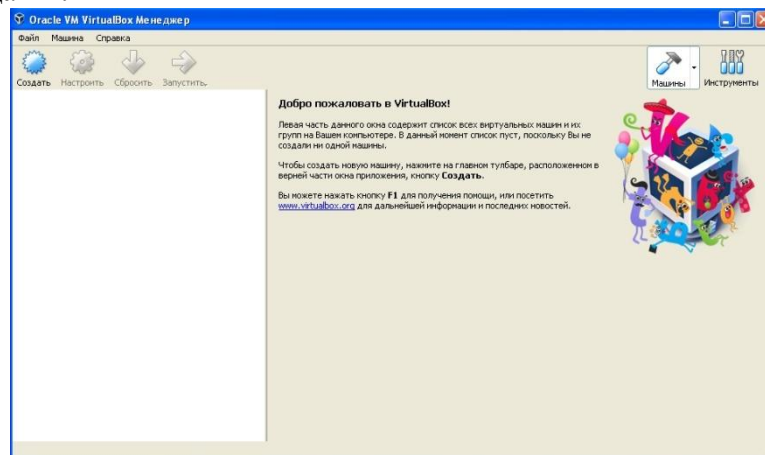
Сохраняем образ установочного диска на жесткий диск.



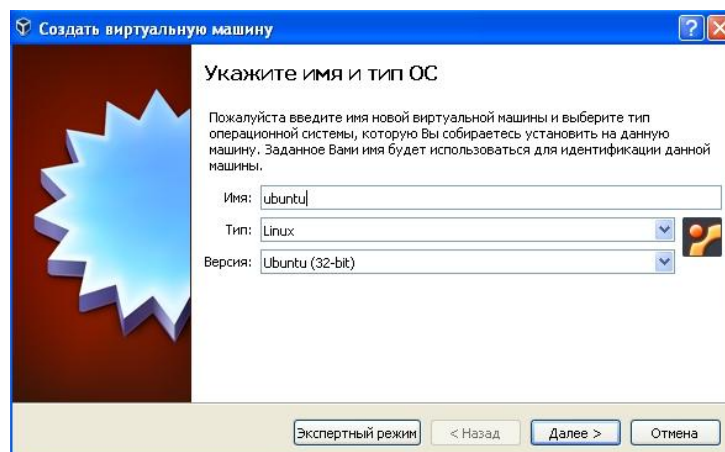
(Альтернативно можно скачать, набрав в адресной строке браузера прямую ссылку на образ установочного диска операционной системы. На момент составления методички последняя стабильная версия операционной системы с долговременной поддержкой имела версию 16.04. Прямая ссылка на образ установочного диска выглядела следующим образом: <http://releases.ubuntu.com/16.04/ubuntu-16.04.3-desktop-i386.iso>).

4. Создать и настроить виртуальную машину

Запускаем установленное приложение OracleVirtualBox. В появившемся окне в левой части перечислены все созданные виртуальные машины. В правой части отображаются свойства выбранной в списке виртуальной машины. Сразу после установки список виртуальных машин пуст. Для создания новой виртуальной машины необходимо в верхней части окна нажать на кнопку «Создать».

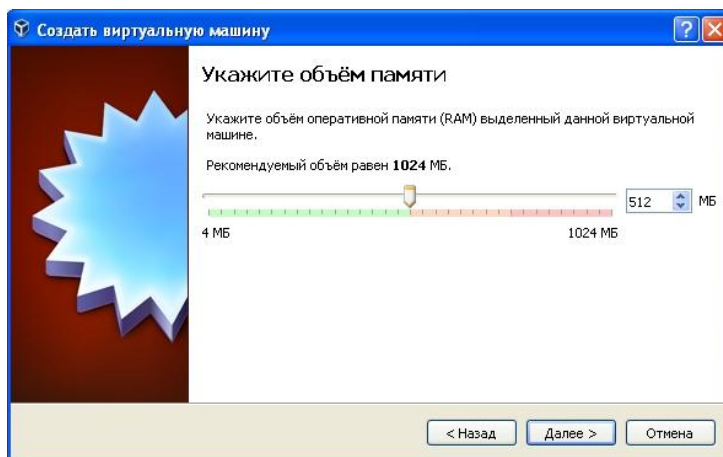


При создании для новой виртуальной машины необходимо задать оригинальное имя, сопоставимое с ФИО студента, тип и версию операционной системы, которая планируется для установки.

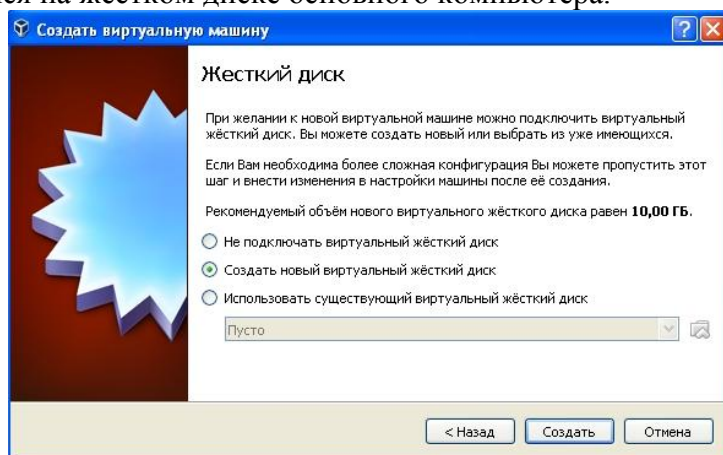


В зависимости от выбранного типа и версии операционной системы виртуальной машины будут установлены остальные параметры в значения, рекомендованные для функционирования. Все эти значения в процессе настройки можно будет изменить.

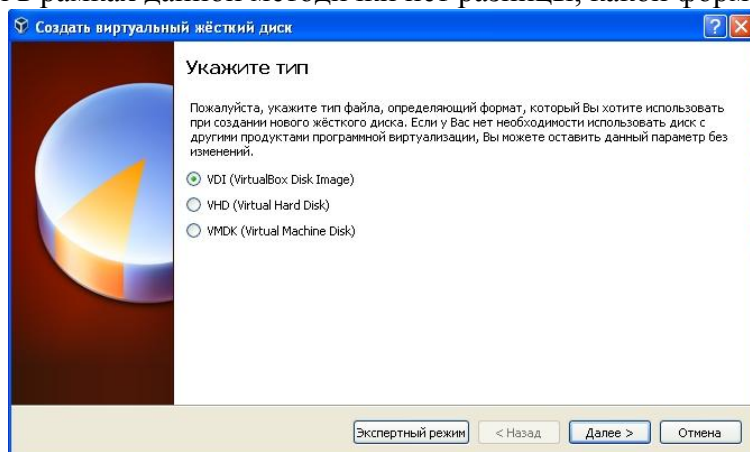
Далее необходимо задать объем оперативной памяти, выделяемой для функционирования виртуальной машины. Внимание! Не выделяйте более 50% объема физической памяти компьютера для виртуальной машины, так как это может привести к нестабильной работе компьютера.



После выделения оперативной памяти для виртуальной машины необходимо задать размер жесткого диска. Жесткий диск виртуальной машины представляет собой один или несколько файлов особого формата, содержащие всю информацию виртуальной машины. Все изменения, вносимые на жесткий диск виртуальной машины, не могут причинить вред информации, хранящейся на жестком диске основного компьютера.

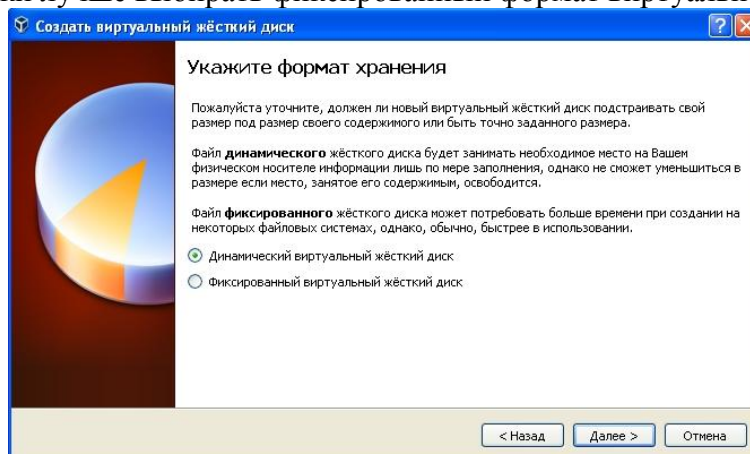


В OracleVirtualBox поддерживается несколько форматов файлов жестких дисков. С точки зрения обучения в рамках данной методички нет разницы, какой формат выбрать.

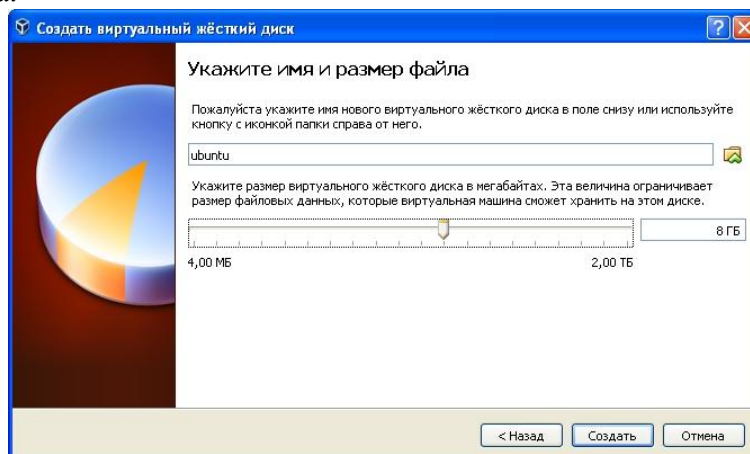


Существует два поддерживаемых формата хранения файлов жесткого диска вирту-

альной машины: динамический и фиксированный. Для экономии места на жестком диске можно выбрать динамический формат хранения данных жесткого диска виртуальной машины. Однако, из практики лучше выбирать фиксированный формат виртуального диска.



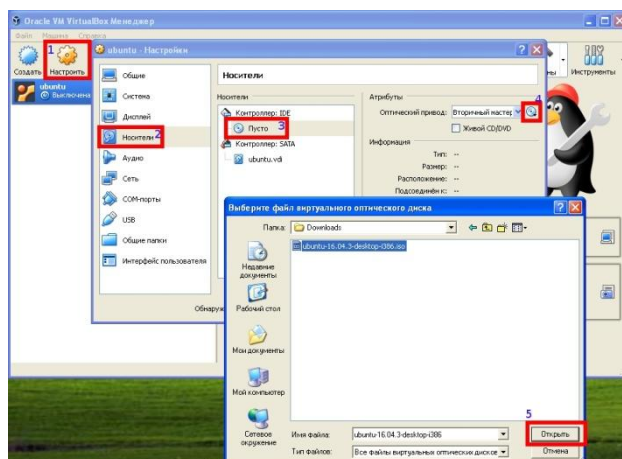
Далее необходимо задать имя файла жесткого диска и максимальный размер файловых данных, который виртуальная машина сможет хранить. Внимание! При выборе фиксированного формата хранения жесткого диска заданный объем сразу будет занят файлом. Таким образом, при задании размера необходимо оценивать требуемый для работы виртуальной машины объем дискового пространства и наличие свободного места на жестком диске основного компьютера.



5. Подготовить виртуальную машину к установке операционной системы

Для установки операционной системы необходимо в виртуальный привод подключить скаченный на 3-м шаге образ установочного диска. Для этого необходимо в списке виртуальных машин выделить созданную нами виртуальную машину и в верхней части окна нажать на кнопку «Настроить».

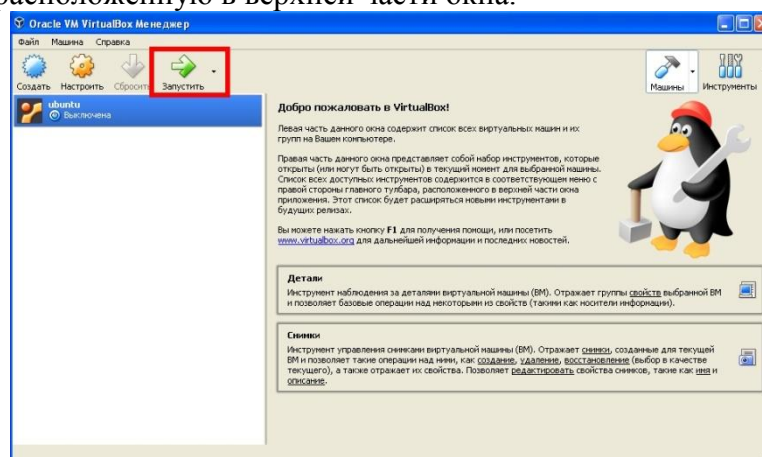
В появившемся окне необходимо перейти во вкладку «Носители», выбрать виртуальный привод и подключить скаченный образ установочного диска.



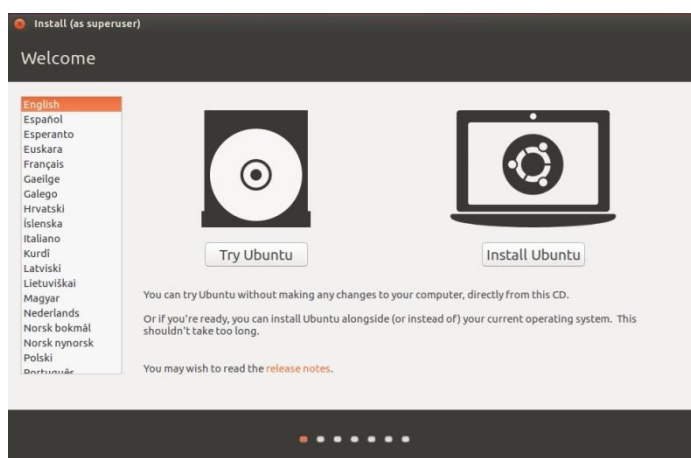
Дополнительно в окне настроек виртуальной машины можно настроить остальные параметры работы.

6. Установить операционную систему Ubuntu Linux на виртуальную машину

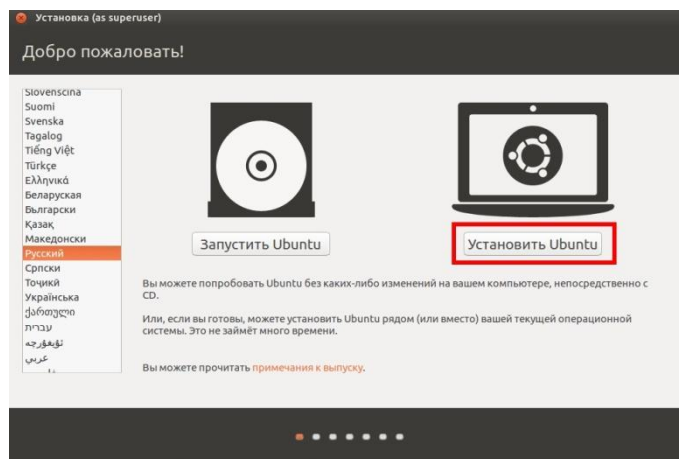
Для начала установки операционной системы необходимо запустить созданную виртуальную машину. Для этого надо выбрать ее в списке виртуальных машин и нажать на кнопку «Запустить», расположенную в верхней части окна.



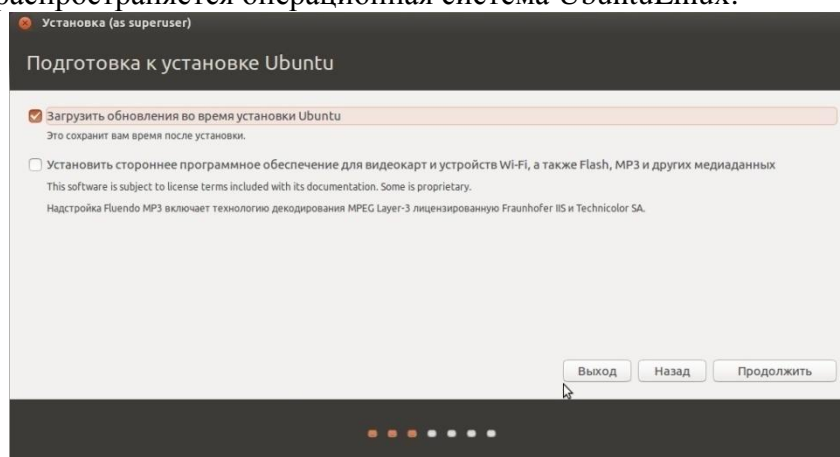
Виртуальная машина начнет загружаться с установочного образа диска, смонтированного в привод. Автоматически будет запущен процесс установки операционной системы. Первым шагом необходимо выбрать язык, который будет использоваться в процессе установки.



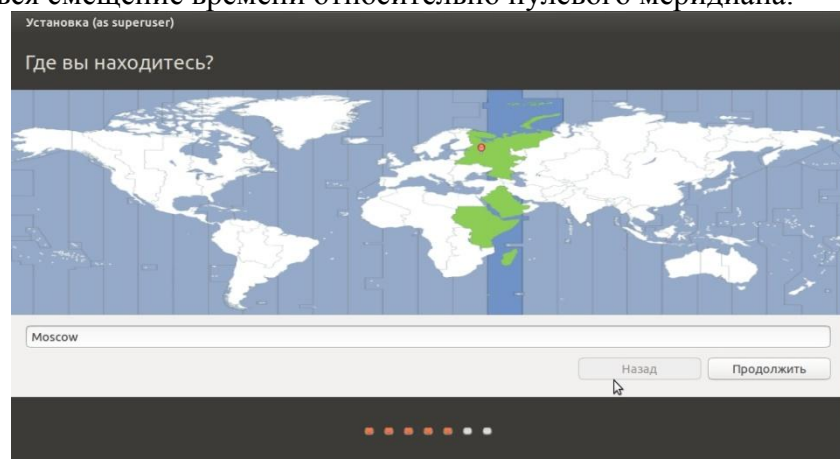
После выбора языка необходимо выбрать один из предложенных сценариев: использовать операционную систему без установки или установить операционную систему на жесткий диск виртуальной машины. Для выполнения задания лабораторной работы необходимо выбрать пункт «Установить Ubuntu».



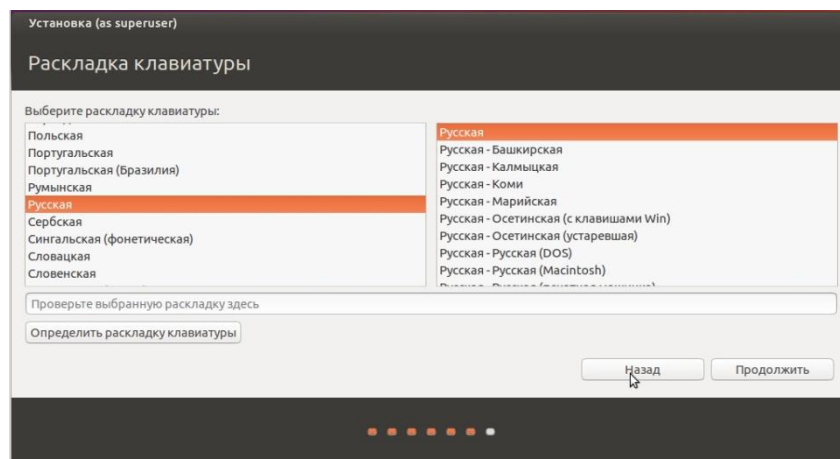
При установке можно затребовать установку последних обновлений компонентов операционной системы, а также установку компонент, использование которых может быть ограничено условиями лицензионных соглашений, не подпадающих под условия лицензии GPL, под которой распространяется операционная система UbuntuLinux.



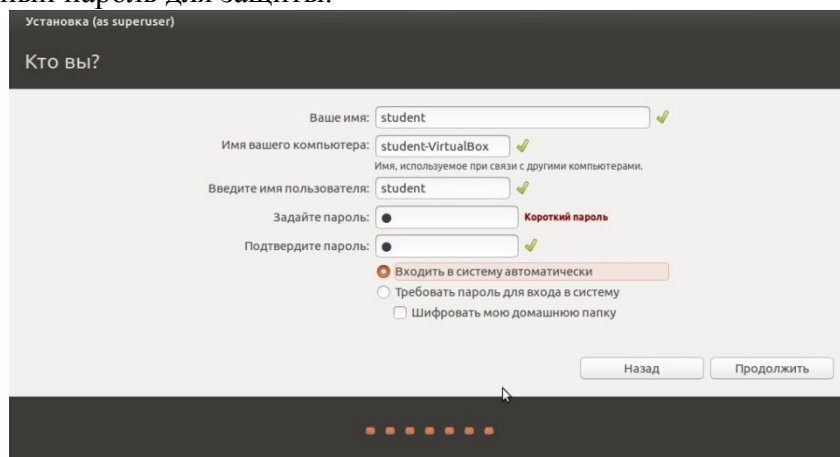
Далее необходимо задать часовой пояс. В зависимости от выбранного часового пояса будет рассчитываться смещение времени относительно нулевого меридиана.



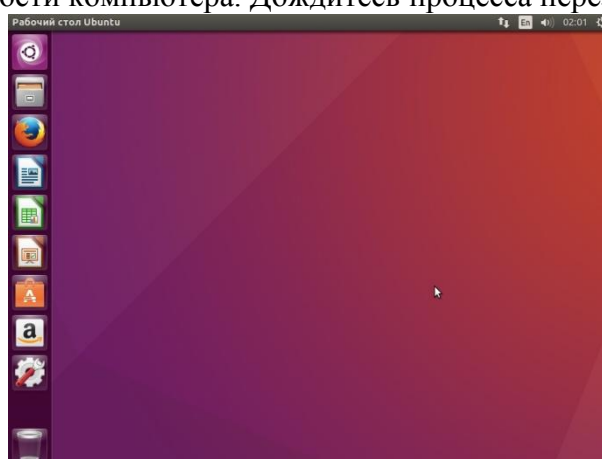
Для комфортной работы с операционной системой необходимо выбрать раскладку клавиатуры, используемую по умолчанию. Дополнительную раскладку клавиатуры можно будет задать после установки операционной системы.



Для представления в системе необходимо создать учетную запись, от имени которой будет осуществляться работа. Учетная запись должна соответствовать идентификации студента, т.к. все работы будут выполняться именно с этим идентификатором. Внимание! Создаваемая на данном этапе учетная запись будет обладать правами администратора системы. Придумайте надежный пароль для защиты.



После задания всех параметров начнется установка операционной системы на жесткий диск виртуальной машины. По завершении установки будет предложено перезагрузить виртуальную машину. Процесс перезагрузки может продолжаться длительное время, в зависимости от производительности компьютера. Дождитесь процесса перезагрузки.



Выводы

В данной лабораторной работе мы приобрели навыки установки OracleVirtualBox, познакомились с процессом создания виртуальной машины. Выполнили установку и начальную настройку операционной системы UbuntuLinux. Созданная в процессе выполнения виртуальная машина понадобится для выполнения последующих лабораторных работ.

Полная установка OracleVirtualBox и операционной системы заняла 1 час 15 минут.

Архитектура операционной системы LINUX.

Система LINUX представляет собой интерактивную программу, разработанную для одновременной поддержки нескольких процессов и нескольких пользователей. LINUX была разработана для опытных пользователей и программистов, поэтому ее основными свойствами являются: непротиворечивость, гибкость и мощь. Это означает, что в системе должно быть небольшое количество базовых элементов, которые можно комбинировать бесконечным числом способов, чтобы приспособить их для конкретного приложения. Кроме того, LINUX лишена избыточности, например, вместо команды `soru` можно написать `sr`. Одно из основных правил LINUX заключается в том, что каждая программа должна выполнять всего одну функцию, но делать это хорошо.

Операционная система LINUX представляет собой иерархическую многоуровневую структуру (рис. 1). На нижнем уровне располагается аппаратное обеспечение, состоящее из центрального процессора, памяти, дисков, терминалов и других устройств. На вышележащем уровне находится операционная система, функция которой заключается в управлении аппаратным обеспечением и предоставлении всем программам интерфейса системных вызовов, которые позволяют программам создавать процессы, файлы и др. ресурсы и управлять ими. Программы обращаются к системным вызовам, помещая аргументы в регистры центрального процессора (иногда в стек) и выполняя команду эмулированного прерывания для переключения из пользовательского режима в режим ядра и передачи управления операционной системе LINUX. На языке C нельзя написать команду эмулированного прерывания, поэтому процедуры написаны на ассемблере, но могут вызываться из программ, написанных на C. Процедуры представляют собой библиотечные функции, по одной на системный вызов.

Каждая такая процедура помещает аргументы в определенное место и выполняет команду эмулированного прерывания TRAP. Чтобы обратиться к системному вызову `read`, программа, написанная на языке C должна вызвать библиотечную процедуру `_read`. Стандарт POSIX определяет библиотечные процедуры, соответствующие системным вызовам, их параметры, их действия и результат выполнения этих действий. LINUX содержит большое количество стандартных программ, некоторые из них описываются стандартом POSIX 1003.2: компиляторы, редакторы, программы обработки текста и утилиты для работы с файлами.

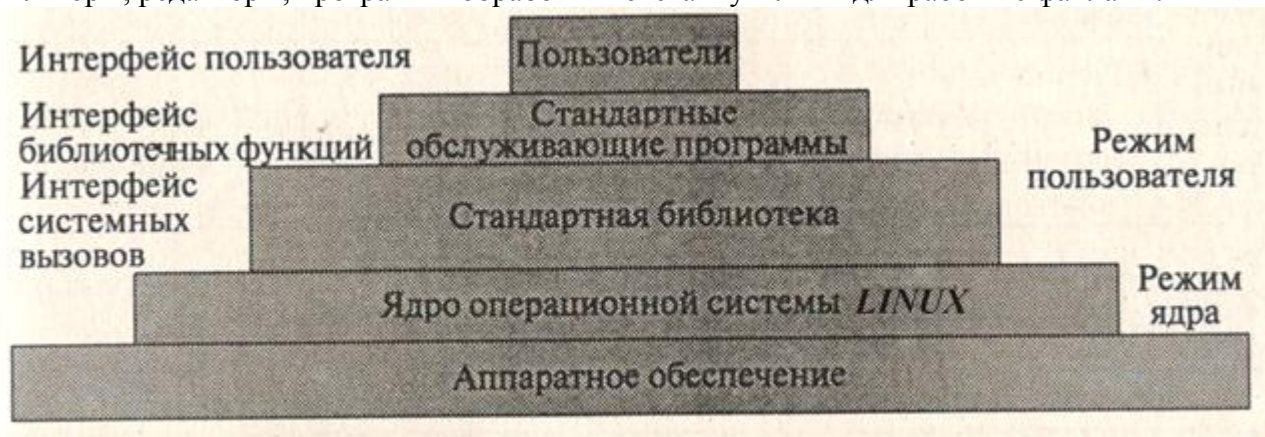


Рис. 1 Уровни операционной системы LINUX

Именно эти программы и запускаются пользователем с терминала. Таким образом, речь идет о трех интерфейсах в операционной системе LINUX: интерфейсе системных вызовов, интерфейсе библиотечных функций и интерфейсе, образованном набором стандартных обслуживающих программ.

Ядро операционной системы LINUX состоит из нескольких уровней (рис. 2). Нижний уровень ядра состоит из драйверов устройств и процедуры диспетчеризации процессов. драйверы системы LINUX подразделяются на два класса: драйверы символьных устройств и драйверы блочных устройств. Основное различие между ними заключается в том, что на блочных устройствах разрешается операция поиска, а на символьных нет.

Технически сетевые устройства представляют собой символьные устройства, но обрабатываются по-иному, поэтому они выделены в отдельный класс. Диспетчеризация процессов производится при возникновении прерывания. При этом низкоуровневая программа останавливает выполнение работающего процесса, сохраняет его состояние в таблице процессов ядра и запускает соответствующий драйвер. Диспетчеризация процессов производится также, когда ядро завершает свою работу и наступает момент запуска процесса пользователя. Программа диспетчеризации процессов написана на ассемблере и представляет собой отдельную от процедуры планирования программу.

Символьные устройства могут использоваться двумя способами. Некоторым программам (например, текстовым редакторам vi и emacs), требуется нажатая клавиша без обработки. Для этого служит ввод-вывод с необработанного терминала (телетайпа). Другое программное обеспечение, например оболочка shell, принимает на входе уже готовую текстовую строку, позволяя пользователю редактировать ее, пока не будет нажата клавиша Enter. Такое программное обеспечение пользуется вводом с терминала в обработанном виде и дисциплинами линии связи.

Системные вызовы					Аппаратные и эмулирован- ные прерывания		
Управление терминалом		Сокеты	Имено- вание файлов	Ото- браже- ние адресов	Стра- ничные преры- вания	Обра- ботка сигна- лов	Создание и завер- шение процес- сов
Необ- рабо- тан- ный теле- тайп	Обрабо- танный телетайп	Сетевые протоколы	Файло- вые сис- темы	Виртуальная па- мять			
	Дисцип- лины линии связи	Маршрути- зация	Буфер- ный кэш	Страничный кэш		Планирование процесса	
Символьные устройства		Драйверы сетевых устройств	Драйверы дисковых уст- ройств			Диспетчеризация процессов	
Аппаратура							

Рис. 2. Структура ядра операционной системы LINUX.

Сетевое программное обеспечение часто бывает модульным, с поддержкой множества различных устройств и протоколов. Уровень выше сетевых драйверов выполняет функции маршрутизации, гарантируя, что правильный пакет направляется правильному устройству или блоку управления протоколами. Большинство систем LINUX содержат в своем ядре полноценный маршрутизатор Интернета. Над уровнем маршрутизации располагается стек протоколов (включая протоколы IP и TCP). Над сетевыми протоколами располагается интерфейс сокетов, позволяющий программам создавать сокеты¹ для отдельных сетей и протоколов. Для использования сокетов пользовательские программы получают дескрипторы файлов.

Над дисковыми драйверами располагаются буферный кэш и страничный кэш файло-

¹ Сокеты подобны почтовым ящикам и телефонным розеткам в том смысле, что они образуют пользовательский интерфейс с сетью, как почтовые ящики формируют интерфейс с почтовой системой, телефонные розетки позволяют абоненту подключать телефон и соединяться с телефонной системой.

вой системы. В ранних системах LINUX буферный кэш представлял собой фиксированную область памяти, а остальная память использовалась для страниц пользователя. В современных системах границы не существует, и любая страница памяти может использоваться для выполнения поставленной задачи. Над буферным кэшем располагаются файловые системы. Большинство систем *LINUX* поддерживают несколько файловых систем, включая быструю файловую систему Беркли, журнальную файловую систему, а также различные виды файловых систем *System V*. Файловые системы совместно используют общий буферный кэш. Выше файловых систем помещается именование файлов, управление каталогами, управление жесткими и символическими связями, а также другие свойства файловой системы, одинаковые для всех файловых систем.

Над страничным кэшем располагается система виртуальной памяти, содержащая логические алгоритмы работы со страницами. Выше находится программа отображения файлов на виртуальную память и высокоуровневая программа управления страничными прерываниями, которая определяет действия при их возникновении. Вначале она проверяет допустимость обращения к памяти. При положительном ответе определяет местонахождение требуемой страницы и способ ее получения. Над диспетчером располагается планировщик процессов, выбирающий процесс, который должен быть запущен следующим. Если потоками управляет ядро, то управление потоками является также функцией ядра.

В некоторых системах LINUX управление потоками вынесено в пространство пользователя. Над планировщиком процессов расположена программа для обработки сигналов и отправки их в требуемом направлении, а также программа, занимающаяся созданием и завершением процессов. Верхний уровень представляет собой интерфейс системы. Слева располагается интерфейс системных вызовов, который принимает все системные вызовы и направляет их одному из модулей низших уровней в зависимости от природы системного вызова. Правая часть верхнего уровня представляет собой вход для аппаратных и эмулированных прерываний, включая сигналы, страничные прерывания, разнообразные исключительные ситуации процессора и прерывания ввода-вывода.

Файловая система: изучение команд работы с файлами и каталогами

Введение

Файловая система— порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов (и каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Цель лабораторной работы

Лабораторная работа выполняется в среде, установленной и настроенной в процессе выполнения лабораторной работы №1 или в среде, установленной в компьютерном классе.

Целью данной лабораторной работы является изучение команд операционной системы GNU Linux по работе с элементами файловой системы, а также получение практических навыков создания, изменения, манипулирования и удаления файлов и каталогов.

В результате выполнения лабораторной работы студенты познакомятся с процессом создания структуры каталогов, изучат различные способы создания и манипулирования данными. На примере созданной в процессе лабораторной работы базы данных на основе текстовых файлов будут рассмотрены вопросы сортировки и фильтрации информации, вывод требуемых данных на экран и в файл.

Основные команды для работы с файловой системой

Для получения подробной справки по каждой из команд необходимо набрать команду `man` “имя команды”. В справке содержится описание команды, область ее применения, синтаксис вызова, возможные параметры вызова.

- `.` – ссылка на текущий каталог. Текущим называется каталог, с которым работает операционная система, если ей не указать другого каталога.
- `..` – ссылка на родительский каталог. Родительским каталогом называется каталог, в котором находится текущий.
- **cat** – команда объединения/слияния данных. Имя команды является сокращением от английского слова concatenate.

`cat filename.txt` – выводит в стандартный поток вывода содержимое файла `filename.txt`. Если после команды указать несколько имен файлов, разделенных символом пробела, содержимое файлов будет объединено в один блок и выведено в стандартный поток вывода.

Есть возможность перенаправить вывод на устройство или в файл, используя оператор `>`. Пример: `cat filename1.txt filename2.txt > filename3.txt`. В данном примере содержимое файлов `filename1.txt` и `filename2.txt` будет объединено и записано в файл `filename3.txt`. При этом если файл `filename3.txt` существовал, он будет перезаписан. Если необходимо дописать информацию в конец файла, необходимо использовать оператор `>>`. Пример: `cat filename1.txt >> filename2.txt`. Данные из файла `filename1.txt` будут дописаны в конец файла `filename2.txt`. Если `filename2.txt` не существовало, он будет создан.

Также команда `cat` используется для организации конвейера для ввода информации с клавиатуры в файл. В этом случае формат команды следующий: `cat > filename.txt << EOF`. По выполнении команды последовательно будет запрашиваться информация с клавиатуры. Для завершения ввода необходимо с новой строки ввести последовательность “EOF”.

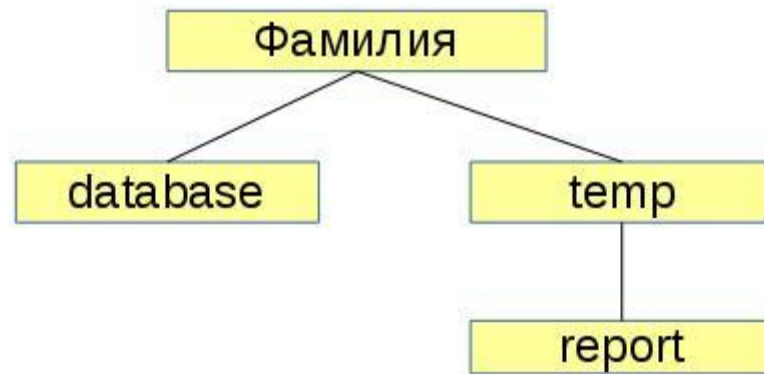
- **cd** – команда для изменения текущего каталога. В качестве аргумента команды задается абсолютное или относительное имя каталога, который необходимо сделать текущим.

щим.

- **echo**— команда, предназначенная для вывода строки текста в стандартный поток вывода. Команда поддерживает возможность перенаправления вывода (см. примеры для команды cat).
- **tree** - команда выводит содержимое текущего каталога в виде дерева.
- **grep** — команда строковой фильтрации текстовых данных. Она использует компактный недетерминированный алгоритм сопоставления. В качестве параметра принимает строку шаблона для поиска, сформированную в соответствии с правилами составления паттернов для регулярных выражений (стандарт PERL). Команда может использоваться как самостоятельно, принимая на вход имя файла, так и в составе конвейера.
- **ls** — команда для вывода в стандартный поток вывода содержимого каталога.
- **mkdir** — команда для создания директории. Для выполнения команды необходимо обладать правами на запись для текущего каталога. Идентификатор владельца и группы нового каталога устанавливаются соответственно равными реальным идентификаторам владельца и группы процесса, в контексте которого выполняется команда.
- **nano** — консольный текстовый редактор для Unix и Unix-подобных операционных систем.
- **pwd** — команда UNIX-подобных системах, которая выводит полный путь от корневого каталога к текущему рабочему каталогу: в контексте которого (по умолчанию) будут исполняться вводимые команды.
- **sort**— команда для сортировки содержимого файла в алфавитном или нумерологическом порядке. Если задать несколько файлов, то команда sort соединит их и, рассортировав, выдаст единым выводом. По умолчанию, объектом сортировки будут строки, однако опции позволяют выбирать объект сортировки: колонки, столбцы и прочие элементы форматирования файла. Разделителем между ними служат пробелы, однако соответствующие опции позволяют задать иные разделители.
- **uniq**— команда, с помощью которой можно вывести или отфильтровать повторяющиеся строки в файле. Если входной файл задан как («-») или не задан вовсе, чтение производится из стандартного потока ввода. Если выходной файл не задан, запись производится в стандартный поток вывода. Вторая и последующие копии повторяющихся соседних строк не записываются. Повторяющиеся входные строки не распознаются, если они не следуют строго друг за другом, поэтому может потребоваться предварительная сортировка файлов.
- **wc**— команда подсчета строк, слов и символов С помощью команды wc можно подсчитать число строк, слов и символов в указанном файле. Если указано более одного файла в командной строке, то команда wc осуществляет подсчет строк, слов и символов в каждом файле и затем выдает общее число.

Порядок выполнения работы

1. Все этапы выполнения работы необходимо фиксировать.
2. Войти в систему от имени своей учетной записи. В случае, если вход осуществлен в графическую оболочку, переключиться на текстовую консоль или запустить терминал. Вся дальнейшая работа выполняется исключительно в терминале. Использование root прав запрещено.
3. Создать родительский каталог. В качестве имени каталога задать свою фамилию. Все остальные действия данной лабораторной работы будут выполняться внутри данного каталога.
4. Внутри каталога, созданного на 2-м шаге создать структуру каталогов, представленную на рисунке. Вывести на экран содержимое текущего каталога и убедиться, что все созданные каталоги созданы без ошибок. Для отображения используйте утилиту tree. При необходимости произведите обновление компонентов и установите утилиту tree вручную.



5. Перейти в каталог temp. Убедиться, что он является текущим. Вывести на экран содержимое каталога.

6. С помощью встроенного текстового редактора внутри каталога temp создать файл базы данных dataset1.txt. Заполнить файл данными в соответствии с номером варианта задания. В качестве разделителя столбцов данных в файле использовать символ “;” без пробелов. Файл должен содержать не менее 3-х строк.

7. С помощью конвейера команд внутри каталога temp создать файл базы данных dataset2.txt. Заполнить файл данными в соответствии с номером варианта задания. В качестве разделителя столбцов данных в файле использовать символ “;” без пробелов. Файл должен содержать не менее 4-х строк. Данные должны отличаться от введенных ранее.

8. С помощью перенаправления вывода в файл, либо используя команду echo создать файл базы данных dataset3.txt. Заполнить файл данными в соответствии с номером варианта задания. В качестве разделителя столбцов данных в файле использовать символ “;” без пробелов. Файл должен содержать не менее 3-х строк. Данные должны отличаться от введенных ранее.

9. Вывести на экран содержимое всех созданных файлов базы данных.

10. Объединить содержимое всех созданных файлов базы данных в один файл data.txt и поместить его в каталог /database.

11. Перейти в каталог /database. Убедиться, что он является текущим. Вывести на экран содержимое каталога. Убедиться, что созданный файл data.txt содержит все необходимые данные.

12. Подсчитать количество строк файла data.txt. Результат подсчета вывести на экран и в файл отчета output.txt, расположенный в каталоге report (см. рисунок).

13. С помощью любого из использованных выше способов дополнить файл data.txt 2-я строками данных в соответствии с номером варианта задания. В качестве разделителя столбцов данных в файле использовать символ “;” без пробелов. Убедиться, что файл data.txt содержит все необходимые данные.

14. Повторно подсчитать количество строк файла data.txt. Результат подсчета вывести на экран и дописать в конец файла отчета output.txt, расположенного в каталоге report.

15. Осуществить фильтрацию данных файла data.txt в соответствии с номером варианта задания. Результат фильтрации вывести на экран и в файл отчета filtered.txt, расположенный в каталоге report. Повторить фильтрацию с различными значениями фильтра. Результаты фильтрации выводить на экран и дописывать в файл отчета filtered.txt.

16. Выполнить сортировку содержимого файла data.txt в соответствии с номером варианта задания. Результат сортировки вывести на экран и в файл отчета sorted.txt, расположенный в каталоге report.

17. Выполнить фильтрацию содержимого файла data.txt с сортировкой результата фильтрации. Фильтрацию и сортировку выполнить в соответствии с номером варианта задания. Результат вывести на экран и в файл отчета filteredsorted.txt, расположенный в каталоге report.

18. Исследовать самостоятельно команды: date, cal, pwd, who, clear, exit.
19. Выполнить команду вывода календаря на экран и любым известным способом записать значение в файл calendar.txt, находящийся в каталоге /database. Результат вывести на экран.

Выводы

В данной лабораторной работе мы приобрели навыки работы с файлами и каталогами, познакомились с некоторыми командами манипулирования данными на примере текстовой базы данных. Рассмотренные в лабораторной работе команды shell могут применяться при выполнении рутинных операций по управлению и обслуживанию операционных систем, а также для автоматизации некоторых задач (администрирование, программирование и т.п.)

Варианты заданий

- 1 Автомобили (ФИО владельца, модель, год выпуска, место регистрации). Поиск по модели автомобиля. Сортировка по году выпуска.
- 2 Библиотека (ФИО автора, название произведения, год издания, издательство). Поиск по издательству. Сортировка по году издания.
- 3 Поликлиника (Номер поликлиники, специалисты, ФИО, дни приема). Поиск по специалистам. Сортировка по дням приема.
- 4 Бухгалтерия (ФИО сотрудников, год поступления на работу, зарплата, номер отдела). Поиск по зарплате. Сортировка по отделам.
- 5 Цветы (название цветка, окраска, месяц цветения, место произрастания). Поиск по названию цветка. Сортировка по месту произрастания.
- 6 Институт (ФИО студента, курс, группа, размер стипендии). Поиск по ФИО. Сортировка по размеру стипендии.
- 7 Преподаватель (ФИО преподавателя, должность, название кафедры, факультет). Поиск по ФИО преподавателя. Сортировка по факультету.
- 8 Спортивная команда (ФИО спортсмена, возраст, рост, вид спорта). Поиск по виду спорта. Сортировка по возрасту.
- 9 Воинская часть (ФИО военнослужащего, звание, подразделение, возраст). Поиск по подразделению. Сортировка по возрасту.
- 10 Экспорт (наименование товара, объем поставки, стоимость единицы продукции, страна экспорта). Поиск по наименованию товара. Сортировка по объемам поставки.
- 11 Телефонный справочник (ФИО абонента, номер телефона, место работы, город). Поиск ФИО. Сортировка по месту работы.
- 12 Авиакомпания (номер рейса, дата вылета, время вылета, пункт назначения). Поиск по пункту назначения. Сортировка по дате вылета.
- 13 Автосервис. (Название, тип выполняемых работ, сроки, цены). Поиск по типу работ. Сортировка цене инверсно.
- 14 Футбольные команды (название команды, ФИО тренера, количество забитых мячей, количество набранных очков). Поиск по названию команды. Сортировка по ФИО тренера.
- 15 Вокзал (номер поезда, тип поезда, количество вагонов, пункт назначения). Поиск по типу поезда. Сортировка по количеству вагонов.
- 16 Квартиросъемщики (ФИО, название улицы, номер дома, номер квартиры). Поиск по названию улицы. Сортировка по номеру дома.

- 17 Порт (название корабля, год постройки, место постройки, тип корабля). Поиск по названию корабля. Сортировка по году постройки.
- 18 Страна (название страны, количество жителей, площадь, столица). Поиск по названию страны. Сортировка по площади инверсно.
- 19 Газета (название газеты, периодичность, тематика, год основания). Поиск по названию тематике. Сортировка по периодичности.
- 20 Фотоаппарат (название фотоаппарата, год выпуска, количество мегапикселей, характеристика зума). Поиск по названию фотоаппарата. Сортировка по количеству мегапикселей.
- 21 Фильмы (название фильма, жанр, год выхода, название студии). Поиск по названию жанру. Сортировка по году выхода инверсно.

Лабораторная работа №3

Использование программируемого фильтра awk

Введение

AWK — это интерпретируемый скриптовый C-подобный язык строчного разбора и обработки входного потока (например, текстового файла) по заданным шаблонам (регулярным выражениям). Используется в bash (SH) скриптах.

Благодаря AWK в нашем распоряжении оказывается язык программирования, а не довольно скромный набор команд, отдаваемых редактору. С помощью языка программирования AWK можно выполнять следующие действия:

- объявлять переменные для хранения данных;
- использовать арифметические и строковые операторы для работы с данными;
- использовать структурные элементы и управляющие конструкции языка, такие, как условные операторы и циклы;
- реализовать сложные алгоритмы обработки данных;
- создавать форматированные отчёты.

AWK может запоминать контекст, делать сравнения, создавать форматированные отчёты, которые удобно читать и анализировать. Это оказывается очень кстати при работе с лог-файлами, которые могут содержать миллионы записей. При надлежащей сноровке, она может объединять множество строк. Awk — это инструмент, предоставляющий несколько очень удобных способов обработки текстовых данных, которые могут пригодиться в повседневной жизни.

Цель лабораторной работы

Лабораторная работа выполняется в среде, установленной и настроенной в процессе выполнения лабораторной работы №1 или в среде, установленной в компьютерном классе.

Целью данной лабораторной работы является изучение возможностей программируемого фильтра AWK при обработке текстовой информации.

В результате выполнения лабораторной работы студенты получают практические навы-

ки манипулирования данными средствами `awk`, составления правил обработки потоков информации, формирования отчетов и извлечения требуемой информации из большого массива данных.

Структура `awk`-программы

В общем виде программа `awk` состоит из операторов (правил), имеющих вид:

шаблон {действие}
шаблон {действие}
...

Шаблон задает правила для отбора обрабатываемых строк в потоке данных. Строки не соответствующие шаблону не обрабатываются. При составлении шаблона используется синтаксис схожий с синтаксисом регулярных выражений языка программирования PERL.

В случае, если шаблон не задан, обрабатываются все строки потока данных.

Например:

```
awk '{print}' f-awk # выдает весь текст;
awk '/до/ {print}' f-awk # выдает строки, где есть "до".
awk '/до/ {}' f-awk # выдает строки, где есть "до"
awk '/до/ {print("Привет!")}' f-awk
```

Действие – последовательность команд манипулирования с данными, заключенная в фигурные скобки. Команды разделяются точкой с запятой, переводом строки или закрывающей скобкой.

Внутри `awk` программы возможны комментарии (как в shell `"#....."`).

Схема вызова `awk`

Программируемый фильтр `awk` может быть вызван из командной строки, командного файла, программы. В общем виде строка вызова `awk` выглядит следующим образом:

`awkoptionsprogramfile`

`Options` – список параметров, позволяющих получить доступ к дополнительным функциям программируемого фильтра. Для просмотра описания всех возможных параметров можно воспользоваться встроенной справкой по команде `awk` (выполнив команду `awk --help`) или вызвать подробное руководство (команда `manawk`).

`Program` – программа `awk`. Программа должна быть заключена в одинарные кавычки.

1. `File` – полный путь к файлу, являющемуся источником данных для программируемого фильтра. Если файл не указан, данные будут браться из стандартного потока ввода/вывода.

`Awk` воспринимает поступающие к нему данные в виде набора записей. Записи представляют собой наборы полей. Упрощенно, если не учитывать возможности настройки `awk` и говорить о некоем вполне обычном тексте, строки которого разделены символами перевода строки, запись — это строка. Поле — это слово в строке

Рассмотрим наиболее часто используемые ключи командной строки `awk`:

- Ffs — позволяет указать символ-разделитель для полей в записи.
- ffile — указывает имя файла, из которого нужно прочесть `awk`-скрипт.
- vvar=value — позволяет объявить переменную и задать её значение по умолчанию,

которое будет использовать `awk`.

-mF`N` — задаёт максимальное число полей для обработки в файле данных.

-mR`N` — задаёт максимальный размер записи в файле данных.

-W`keyword` — позволяет задать режим совместимости или уровень выдачи предупреждений `awk`.

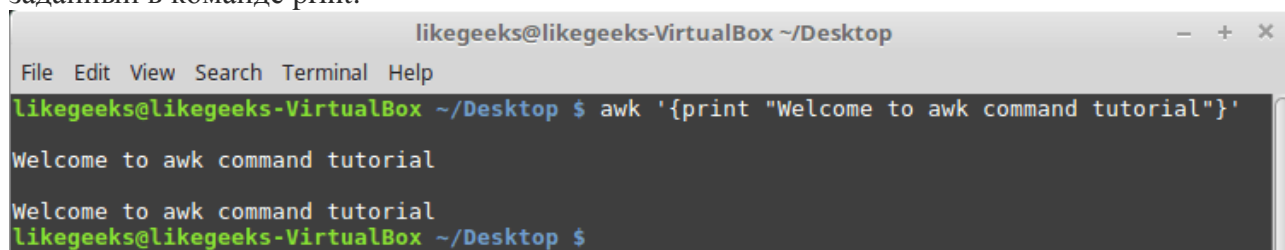
Чтение `awk`-скриптов из командной строки

Скрипты `awk`, которые можно писать прямо в командной строке, оформляются в виде текстов команд, заключённых в фигурные скобки. Кроме того, так как `awk` предполагает, что скрипт представляет собой текстовую строку, его нужно заключить в одинарные кавычки:

```
awk '{print "Welcome to awk command tutorial"}'
```

Запустим эту команду... И ничего не произойдёт. Дело тут в том, что мы, при вызове `awk`, не указали файл с данными. В подобной ситуации `awk` ожидает поступления данных из `STDIN`. Поэтому выполнение такой команды не приводит к немедленно наблюдаемым эффектам, но это не значит, что `awk` не работает — он ждёт входных данных из `STDIN`.

Если теперь ввести что-нибудь в консоль и нажать `Enter`, `awk` обработает введённые данные с помощью скрипта, заданного при его запуске. `Awk` обрабатывает текст из потока ввода построчно, этим он похож на команду `sed`. В нашем случае `awk` ничего не делает с данными, он лишь, в ответ на каждую новую полученную им строку, выводит на экран текст, заданный в команде `print`.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the command 'awk '{print "Welcome to awk command tutorial"}'' being entered and executed. The output 'Welcome to awk command tutorial' is displayed twice, once for each line of input. The prompt 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' is visible at the bottom.

Первый запуск `awk`, вывод на экран заданного текста

Что бы мы ни ввели, результат в данном случае будет одним и тем же — вывод текста. Для того, чтобы завершить работу `awk`, нужно передать ему символ конца файла (EOF, End-of-File). Сделать это можно, воспользовавшись сочетанием клавиш `CTRL + D`.

Позиционные переменные, хранящие данные полей

Одна из основных функций `awk` заключается в возможности манипулировать данными в текстовых файлах. Делается это путём автоматического назначения переменной каждому элементу в строке. По умолчанию `awk` назначает следующие переменные каждому полю данных, обнаруженному им в записи:

- `$0` — представляет всю строку текста (запись).
- `$1` — первое поле.
- `$2` — второе поле.
- `$n` — `n`-ное поле.

Поля выделяются из текста с использованием символа-разделителя. По умолчанию — это пробельные символы вроде пробела или символа табуляции.

Рассмотрим использование этих переменных на простом примере. А именно, обработаем файл, в котором содержится несколько строк (этот файл показан на рисунке ниже) с помощью такой команды:

```
awk '{print $1}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat myfile
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{print $1}' myfile
This
This
This
This
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Вывод в консоль первого поля каждой строки

Здесь использована переменная \$1, которая позволяет получить доступ к первому полю каждой строки и вывести его на экран.

Иногда в некоторых файлах в качестве разделителей полей используется что-то, отличающееся от пробелов или символов табуляции. Выше мы упоминали ключ awk -F, который позволяет задать необходимый для обработки конкретного файла разделитель:

awk -F: '{print \$1}' /etc/passwd

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -F: '{print $1}' /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
```

Указание символа-разделителя при вызове awk

Эта команда выводит первые элементы строк, содержащихся в файле /etc/passwd. Так как в этом файле в качестве разделителей используются двоеточия, именно этот символ был передан awk после ключа -F.

Использование нескольких команд

Вызов awk с одной командой обработки текста — подход очень ограниченный. Awk позволяет обрабатывать данные с использованием многострочных скриптов. Для того, чтобы передать awk многострочную команду при вызове его из консоли, нужно разделить её части точкой с запятой:

```
echo "My name is Tom" | awk '{$4="Adam"; print $0}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo "My name is Tom" | awk '{$4="Adam"; print $0}'
My name is Adam
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Вызов awk из командной строки с передачей ему многострочного скрипта

В данном примере первая команда записывает новое значение в переменную \$4, а вторая выводит на экран всю строку.

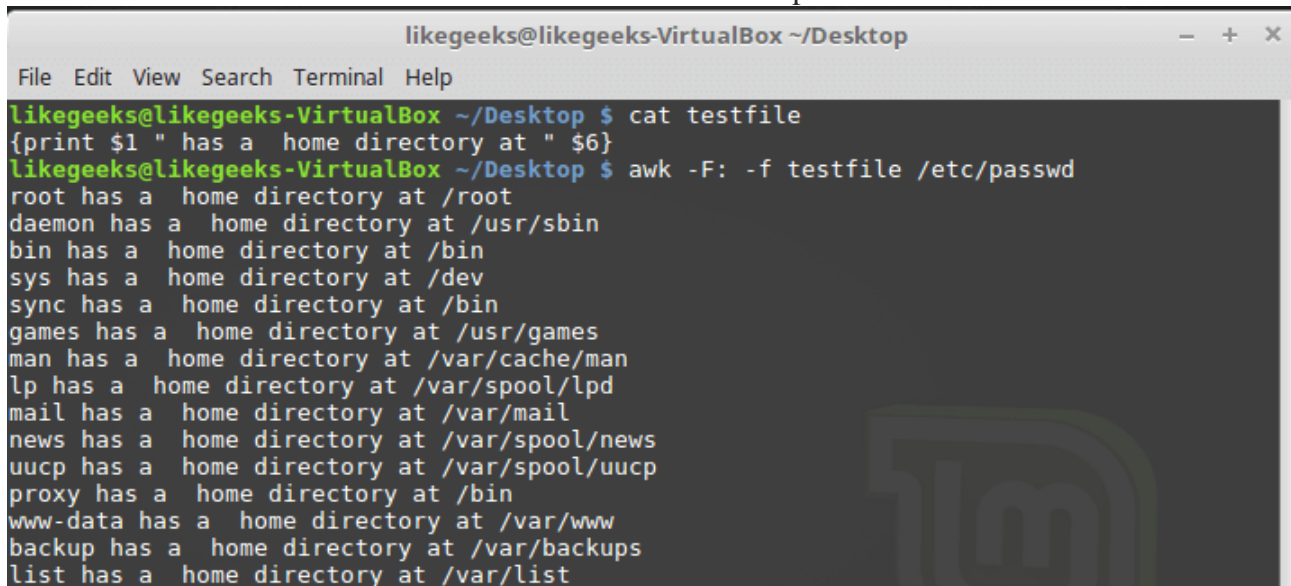
Чтение скрипта awk из файла

Awk позволяет хранить скрипты в файлах и ссылаться на них, используя ключ `-f`. Подготовим файл `testfile`, в который запишем следующее:

```
{print $1 " has a home directory at " $6}
```

Вызовем `awk`, указав этот файл в качестве источника команд:

```
awk -F: -f testfile /etc/passwd
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat testfile
{print $1 " has a home directory at " $6}
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -F: -f testfile /etc/passwd
root has a home directory at /root
daemon has a home directory at /usr/sbin
bin has a home directory at /bin
sys has a home directory at /dev
sync has a home directory at /bin
games has a home directory at /usr/games
man has a home directory at /var/cache/man
lp has a home directory at /var/spool/lpd
mail has a home directory at /var/mail
news has a home directory at /var/spool/news
uucp has a home directory at /var/spool/uucp
proxy has a home directory at /bin
www-data has a home directory at /var/www
backup has a home directory at /var/backups
list has a home directory at /var/list
```

Вызов awk с указанием файла скрипта

Тут мы выводим из файла `/etc/passwd` имена пользователей, которые попадают в переменную `$1`, и их домашние директории, которые попадают в `$6`. Обратите внимание на то, что файл скрипта задают с помощью ключа `-f`, а разделитель полей, двоеточие в нашем случае, с помощью ключа `-F`

В файле скрипта может содержаться множество команд, при этом каждую из них достаточно записывать с новой строки, ставить после каждой точку с запятой не требуется.

Вот как это может выглядеть:

```
{
    text = " has a home directory at "
    print $1 text $6
}
```

Тут мы храним текст, используемый при выводе данных, полученных из каждой строки обрабатываемого файла, в переменной, и используем эту переменную в команде `print`. Если воспроизвести предыдущий пример, записав этот код в файл `testfile`, выведено будет то же самое.

Выполнение команд до начала обработки данных

Иногда нужно выполнить какие-то действия до того, как скрипт начнёт обработку записей из входного потока. Например — создать шапку отчёта или что-то подобное.

Для этого можно воспользоваться ключевым словом `BEGIN`. Команды, которые следуют за `BEGIN`, будут исполнены до начала обработки данных. В простейшем виде это выглядит так:

```
awk 'BEGIN {print "HelloWorld!"}'
```

А вот — немного более сложный пример:

```
awk 'BEGIN {print "The File Contents:"}
{print $0}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN {print "The File Contents:"}
> {print $0}' myfile
The File Contents:
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Выполнение команд до начала обработки данных

Сначала `awk` исполняет блок `BEGIN`, после чего выполняется обработка данных. Будьте внимательны с одинарными кавычками, используя подобные конструкции в командной строке. Обратите внимание на то, что и блок `BEGIN`, и команды обработки потока, являются в представлении `awk` одной строкой. Первая одинарная кавычка, ограничивающая эту строку, стоит перед `BEGIN`. Вторая — после закрывающей фигурной скобки команды обработки данных.

Выполнение команд после окончания обработки данных

Ключевое слово `END` позволяет задавать команды, которые надо выполнить после окончания обработки данных:

```
awk 'BEGIN {print "The File Contents:"}
{print $0}
END {print "End of File"}' myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN {print "The File Contents:"}
> {print $0}
> END {print "End of File"}' myfile
The File Contents:
This is a test.
This is the second test.
This is the thrid test.
This is the fourth test.
End of File
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Результаты работы скрипта, в котором имеются блоки `BEGIN` и `END`

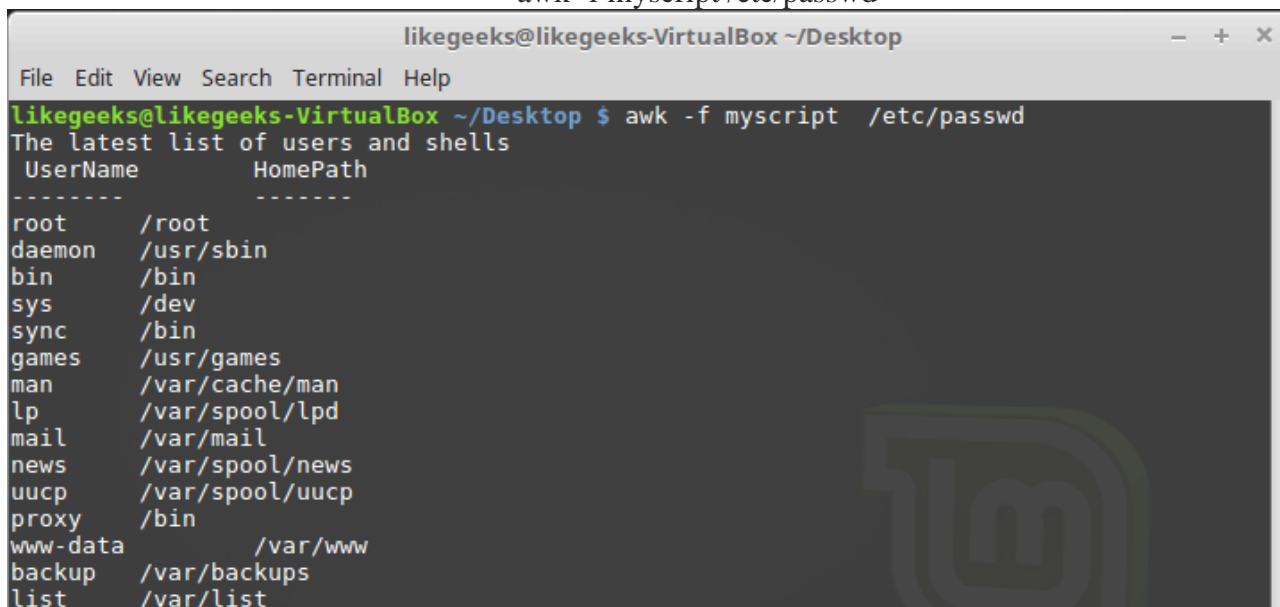
После завершения вывода содержимого файла, `awk` выполняет команды блока `END`. Это полезная возможность, с её помощью, например, можно сформировать подвал отчёта. Теперь напомним скрипт следующего содержания и сохраним его в файле `myscript`:

```
BEGIN {
    print "The latest list of users and shells"
    print " Username \t HomePath"
    print "----- \t -----"
    FS=":"
}
{
    print $1 " \t " $6
}
END {
    print "The end"
}
```

Тут, в блоке `BEGIN`, создаётся заголовок табличного отчёта. В этом же разделе мы указываем символ-разделитель. После окончания обработки файла, благодаря блоку `END`, система сообщит нам о том, что работа окончена.

Запустим скрипт:

```
awk -f myscript /etc/passwd
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk -f myscript /etc/passwd
The latest list of users and shells
  UserName      HomePath
  -----
root           /root
daemon         /usr/sbin
bin            /bin
sys           /dev
sync          /bin
games         /usr/games
man           /var/cache/man
lp            /var/spool/lpd
mail          /var/mail
news         /var/spool/news
uucp         /var/spool/uucp
proxy         /bin
www-data      /var/www
backup        /var/backups
list          /var/list
```

Обработка файла /etc/passwd с помощью awk-скрипта

Встроенные переменные: настройка процесса обработки данных

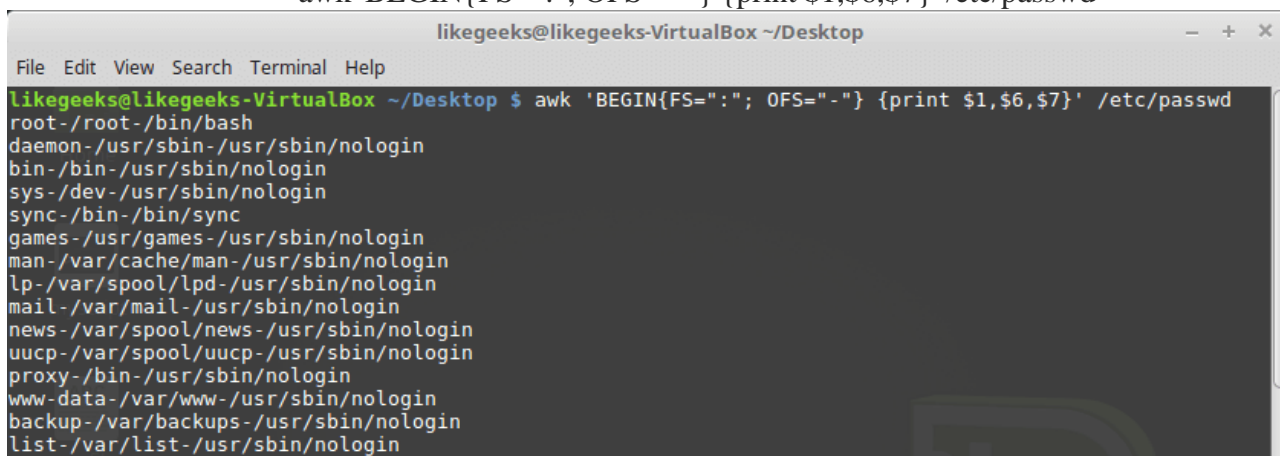
Утилита `awk` использует встроенные переменные, которые позволяют настраивать процесс обработки данных и дают доступ, как к обрабатываемым данным, так и к некоторым сведениям о них.

Мы уже рассматривали позиционные переменные — `$1`, `$2`, `$3`, которые позволяют извлекать значения полей, работали мы и с некоторыми другими переменными. На самом деле, их довольно много. Вот некоторые из наиболее часто используемых:

- `FIELDWIDTHS` — разделённый пробелами список чисел, определяющий точную ширину каждого поля данных с учётом разделителей полей;
- `FS` — уже знакомая вам переменная, позволяющая задавать символ-разделитель полей;
- `RS` — переменная, которая позволяет задавать символ-разделитель записей;
- `OFS` — разделитель полей на выводе `awk`-скрипта;
- `ORS` — разделитель записей на выводе `awk`-скрипта.

По умолчанию переменная `OFS` настроена на использование пробела. Её можно установить так, как нужно для целей вывода данных:

```
awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}' /etc/passwd
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS=":"; OFS="-"} {print $1,$6,$7}' /etc/passwd
root-/root-/bin/bash
daemon-/usr/sbin-/usr/sbin/nologin
bin-/bin-/usr/sbin/nologin
sys-/dev-/usr/sbin/nologin
sync-/bin-/bin/sync
games-/usr/games-/usr/sbin/nologin
man-/var/cache/man-/usr/sbin/nologin
lp-/var/spool/lpd-/usr/sbin/nologin
mail-/var/mail-/usr/sbin/nologin
news-/var/spool/news-/usr/sbin/nologin
uucp-/var/spool/uucp-/usr/sbin/nologin
proxy-/bin-/usr/sbin/nologin
www-data-/var/www-/usr/sbin/nologin
backup-/var/backups-/usr/sbin/nologin
list-/var/list-/usr/sbin/nologin
```

Установка разделителя полей выходного потока

Переменная `FIELDWIDTHS` позволяет читать записи без использования символа-

разделителя полей.

В некоторых случаях, вместо использования разделителя полей, данные в пределах записей расположены в колонках постоянной ширины. В подобных случаях необходимо задать переменную `FIELDWIDTHS` таким образом, чтобы её содержимое соответствовало особенностям представления данных.

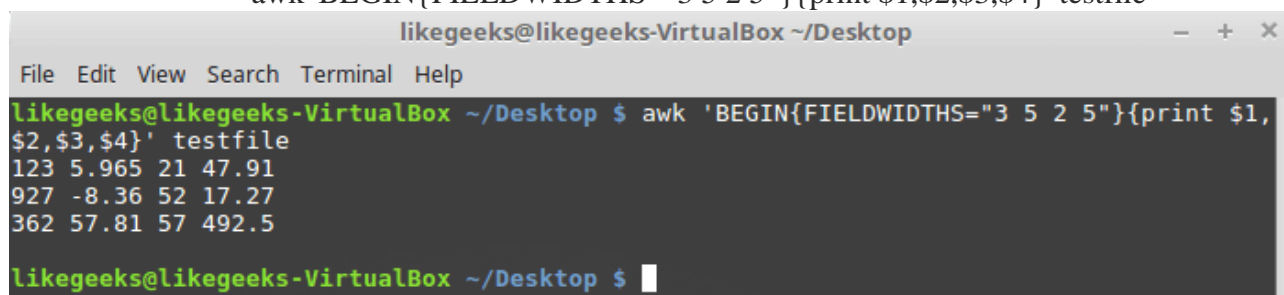
При установленной переменной `FIELDWIDTHS` `awk` будет игнорировать переменную `FS` и находить поля данных в соответствии со сведениями об их ширине, заданными в `FIELDWIDTHS`.

Предположим, имеется файл `testfile`, содержащий такие данные:

```
1235.9652147.91
927-8.365217.27
36257.8157492.5
```

Известно, что внутренняя организация этих данных соответствует шаблону 3-5-2-5, то есть, первое поле имеет ширину 3 символа, второе — 5, и так далее. Вот скрипт, который позволит разобрать такие записи:

```
awk 'BEGIN{FIELDWIDTHS="3 5 2 5"}{print $1,$2,$3,$4}' testfile
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FIELDWIDTHS="3 5 2 5"}{print $1,
$2,$3,$4}' testfile
123 5.965 21 47.91
927 -8.36 52 17.27
362 57.81 57 492.5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Использование переменной `FIELDWIDTHS`

Посмотрим на то, что выведет скрипт. Данные разобраны с учётом значения переменной `FIELDWIDTHS`, в результате числа и другие символы в строках разбиты в соответствии с заданной шириной полей.

Переменные `RS` и `ORS` задают порядок обработки записей. По умолчанию `RS` и `ORS` установлены на символ перевода строки. Это означает, что `awk` воспринимает каждую новую строку текста как новую запись и выводит каждую запись с новой строки.

Иногда случается так, что поля в потоке данных распределены по нескольким строкам. Например, пусть имеется такой файл с именем `addresses`:

```
Person Name
123 High Street
(222) 466-1234
```

```
Another person
487 HighStreet
(523) 643-8754
```

Если попытаться прочесть эти данные при условии, что `FS` и `RS` установлены в значения по умолчанию, `awk` сочтёт каждую новую строку отдельной записью и выделит поля, опираясь на пробелы. Это не то, что нам в данном случае нужно.

Для того, чтобы решить эту проблему, в `FS` надо записать символ перевода строки. Это укажет `awk` на то, что каждая строка в потоке данных является отдельным полем.

Кроме того, в данном примере понадобится записать в переменную `RS` пустую строку. Обратите внимание на то, что в файле блоки данных о разных людях разделены пустой строкой. В результате `awk` будет считать пустые строки разделителями записей. Вот как всё это сделать:

```
awk 'BEGIN{FS="\n"; RS="" } {print $1,$3}' addresses
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS="\n"; RS=""} {print $1, $3}' addresses
Person Name (222) 466-1234
Another person (523) 643-8754
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Результаты настройки переменных RS и FS

Как видите, awk, благодаря таким настройкам переменных, воспринимает строки из файла как поля, а разделителями записей становятся пустые строки.

Встроенные переменные: сведения о данных и об окружении

Помимо встроенных переменных, о которых мы уже говорили, существуют и другие, которые предоставляют сведения о данных и об окружении, в котором работает awk:

- ARGV — количество аргументов командной строки;
- ARGV — массив с аргументами командной строки;
- ARGIND — индекс текущего обрабатываемого файла в массиве ARGV;
- ENVIRON — ассоциативный массив с переменными окружения и их значениями;
- ERRNO — код системной ошибки, которая может возникнуть при чтении или закрытии входных файлов;
- FILENAME — имя входного файла с данными;
- FNR — номер текущей записи в файле данных;
- IGNORECASE — если эта переменная установлена в ненулевое значение, при обработке игнорируется регистр символов;
- NF — общее число полей данных в текущей записи;
- NR — общее число обработанных записей.

Переменные ARGV и ARGV позволяют работать с аргументами командной строки. При этом скрипт, переданный awk, не попадает в массив аргументов ARGV. Напишем такой скрипт:

```
awk 'BEGIN{print ARGV, ARGV[1]}' myfile
```

После его запуска можно узнать, что общее число аргументов командной строки — 2, а под индексом 1 в массиве ARGV записано имя обрабатываемого файла. В элементе массива с индексом 0 в данном случае будет «awk».

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{print ARGV, ARGV[1]}' myfile
2 myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Работа с параметрами командной строки

Переменная ENVIRON представляет собой ассоциативный массив с переменными среды. Опробуем её:

```
awk '
BEGIN{
print ENVIRON["HOME"]
print ENVIRON["PATH"]
}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '
> BEGIN{
> print ENVIRON["HOME"]
> print ENVIRON["PATH"]
> }'
/home/likegeeks
/home/likegeeks/bin:/home/likegeeks/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Работа с переменными среды

Переменные среды можно использовать и без обращения к ENVIRON. Сделать это, например, можно так:

```
echo | awk -v home=$HOME '{print "My home is " home}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ echo | awk -v home=$HOME '{print "My home is " home}'
My home is /home/likegeeks
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Работа с переменными среды без использования ENVIRON

Переменная NF позволяет обращаться к последнему полю данных в записи, не зная его точной позиции:

```
awk 'BEGIN{FS=":"; OFS=":"} {print $1,$NF}' /etc/passwd
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS=":"; OFS=":"} {print $1,$NF}' /etc/passwd
root:/bin/bash
daemon:/usr/sbin/nologin
bin:/usr/sbin/nologin
sys:/usr/sbin/nologin
sync:/bin/sync
games:/usr/sbin/nologin
man:/usr/sbin/nologin
lp:/usr/sbin/nologin
mail:/usr/sbin/nologin
news:/usr/sbin/nologin
uucp:/usr/sbin/nologin
proxy:/usr/sbin/nologin
www-data:/usr/sbin/nologin
backup:/usr/sbin/nologin
list:/usr/sbin/nologin
```

Пример использования переменной NF

Эта переменная содержит числовой индекс последнего поля данных в записи. Обратиться к данному полю можно, поместив перед NF знак \$.

Переменные FNR и NR, хотя и могут показаться похожими, на самом деле различаются. Так, переменная FNR хранит число записей, обработанных в текущем файле. Переменная NR хранит общее число обработанных записей. Рассмотрим пару примеров, передав awk один и тот же файл дважды:

```
awk 'BEGIN{FS=","} {print $1,"FNR="FNR}' myfile myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{FS=","}{print $1,"FNR="FNR}' my
file myfile
This is a test. FNR=1
This is the second test. FNR=2
This is the thrid test. FNR=3
This is the fourth test. FNR=4
This is a test. FNR=1
This is the second test. FNR=2
This is the thrid test. FNR=3
This is the fourth test. FNR=4
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Исследование переменной FNR

Передача одного и того же файла дважды равносильна передаче двух разных файлов. Обратите внимание на то, что FNR сбрасывается в начале обработки каждого файла.

Взглянем теперь на то, как ведёт себя в подобной ситуации переменная NR:

```
awk '
BEGIN {FS=","}
{print $1,"FNR="FNR,"NR="NR}
END{print "There were",NR,"records processed"}' myfile myfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '
> BEGIN {FS=","}
> {print $1,"FNR="FNR,"NR="NR}
> END{print "There were",NR,"records processed"}' myfile myfile
This is a test. FNR=1 NR=1
This is the second test. FNR=2 NR=2
This is the thrid test. FNR=3 NR=3
This is the fourth test. FNR=4 NR=4
This is a test. FNR=1 NR=5
This is the second test. FNR=2 NR=6
This is the thrid test. FNR=3 NR=7
This is the fourth test. FNR=4 NR=8
There were 8 records processed
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Различие переменных NR и FNR

Как видно, FNR, как и в предыдущем примере, сбрасывается в начале обработки каждого файла, а вот NR, при переходе к следующему файлу, сохраняет значение.

Пользовательские переменные

Как и любые другие языки программирования, awk позволяет программисту объявлять переменные. Имена переменных могут включать в себя буквы, цифры, символы подчёркивания. Однако, они не могут начинаться с цифры. Объявить переменную, присвоить ей значение и воспользоваться ей в коде можно так:

```
awk '
BEGIN{
test="This is a test"
print test
}'
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '
> BEGIN{
> test="This is a test"
> print test
> }'
This is a test
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Работа с пользовательской переменной

Условный оператор

Awk поддерживает стандартный во многих языках программирования формат условного оператора if-then-else. Однострочный вариант оператора представляет собой ключевое слово if, за которым, в скобках, записывают проверяемое выражение, а затем — команду, которую нужно выполнить, если выражение истинно.

Например, есть такой файл с именем testfile:

```
10
15
6
33
45
```

Напишем скрипт, который выводит числа из этого файла, большие 20:

```
awk '{if ($1 > 20) print $1}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{if ($1 > 20) print $1}' testfile
33
45
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Однострочный оператор if

Если нужно выполнить в блоке if несколько операторов, их нужно заключить в фигурные скобки:

```
awk '{
if ($1 > 20)
{
x = $1 * 2
print x
}
}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> if ($1 > 20)
> {
> x = $1 * 2
> print x
> }
> }' testfile
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Выполнение нескольких команд в блоке if

Как уже было сказано, условный оператор awk может содержать блок else:

```
awk '{
if ($1 > 20)
{
x = $1 * 2
print x
} else
{
x = $1 / 2
print x
}}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> if ($1 > 20)
> {
> x = $1 * 2
> print x
> } else
> {
> x = $1 / 2
> print x
> }}' testfile
5
7.5
3
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Условный оператор с блоком else

Ветвь else может быть частью однострочной записи условного оператора, включая в себя лишь одну строку с командой. В подобном случае после ветви if, сразу перед else, надо поставить точку с запятой:

```
awk '{if ($1 > 20) print $1 * 2; else print $1 / 2}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{if ($1 > 20) print $1 * 2; else prin
t $1 / 2}' testfile
5
7.5
3
66
90
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Условный оператор, содержащий ветви if и else, записанный в одну строку

Цикл while

Цикл while позволяет перебирать наборы данных, проверяя условие, которое остановит цикл. Вот файл myfile, обработку которого мы хотим организовать с помощью цикла:

```
124 127 130
112 142 135
175 158 245
```

Напишем такой скрипт:

```
awk '{
total = 0
i = 1
while (i < 4)
```

```

{
total += $i
i++
}
avg = total / 3
print "Average:",avg
}' testfile

```

```

likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> i = 1
> while (i < 4)
> {
> total += $i
> i++
> }
> avg = total / 3
> print "Average:",avg
> }' testfile
Average: 127
Average: 129.667
Average: 192.667
likegeeks@likegeeks-VirtualBox ~/Desktop $

```

Обработка данных в цикле while

Цикл `while` перебирает поля каждой записи, накапливая их сумму в переменной `total` и увеличивая в каждой итерации на 1 переменную-счётчик `i`. Когда `i` достигнет 4, условие на входе в цикл окажется ложным и цикл завершится, после чего будут выполнены остальные команды — подсчёт среднего значения для числовых полей текущей записи и вывод найденного значения.

В циклах `while` можно использовать команды `break` и `continue`. Первая позволяет досрочно завершить цикл и приступить к выполнению команд, расположенных после него. Вторая позволяет, не завершая до конца текущую итерацию, перейти к следующей.

Вот как работает команда `break`:

```

awk '{
total = 0
i = 1
while (i < 4)
{
total += $i
if (i == 2)
break
i++
}
avg = total / 2
print "The average of the first two elements is:",avg
}' testfile

```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> i = 1
> while (i < 4)
> {
> total += $i
> if (i == 2)
> break
> i++
> }
> avg = total / 2
> print "The average of the first two elements is:",avg
> }' testfile
The average of the first two elements is: 125.5
The average of the first two elements is: 127
The average of the first two elements is: 166.5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Команда break в цикле while

Цикл for

Циклы for используются во множестве языков программирования. Поддерживает их и awk. Решим задачу расчёта среднего значения числовых полей с использованием такого цикла:

```
awk '{
total = 0
for (i = 1; i < 4; i++)
{
total += $i
}
avg = total / 3
print "Average:",avg
}' testfile
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk '{
> total = 0
> for (i = 1; i < 4; i++)
> {
> total += $i
> }
> avg = total / 3
> print "Average:",avg
> }' testfile
Average: 127
Average: 129.667
Average: 192.667
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Цикл for

Начальное значение переменной-счётчика и правило её изменения в каждой итерации, а также условие прекращения цикла, задаются в начале цикла, в круглых скобках. В итоге нам не нужно, в отличие от случая с циклом while, самостоятельно инкрементировать счётчик.

Форматированный вывод данных

Команда `printf` в `awk` позволяет выводить форматированные данные. Она даёт возможность настраивать внешний вид выводимых данных благодаря использованию шаблонов, в которых могут содержаться текстовые данные и спецификаторы форматирования.

Спецификатор форматирования — это специальный символ, который задаёт тип выводимых данных и то, как именно их нужно выводить. `Awk` использует спецификаторы форматирования как указатели мест вставки данных из переменных, передаваемых `printf`.

Первый спецификатор соответствует первой переменной, второй спецификатор — второй, и так далее.

Спецификаторы форматирования записывают в таком виде:

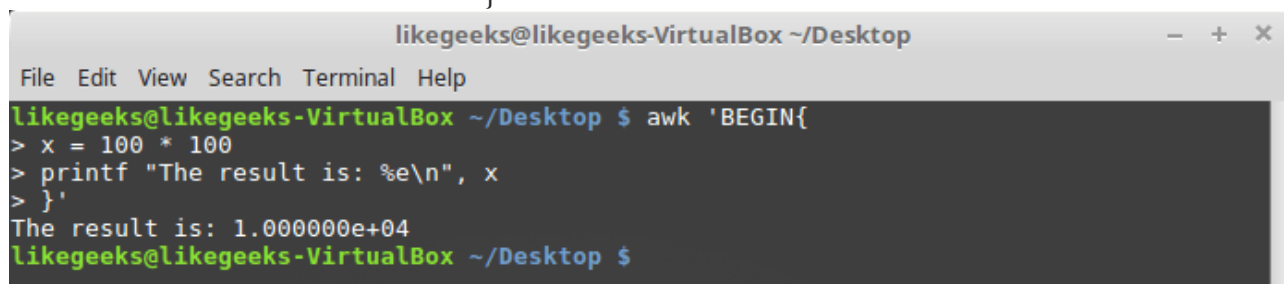
`%[modifier]control-letter`

Вот некоторые из них:

- `c` — воспринимает переданное ему число как код ASCII-символа и выводит этот символ;
- `d` — выводит десятичное целое число;
- `i` — то же самое, что и `d`;
- `e` — выводит число в экспоненциальной форме;
- `f` — выводит число с плавающей запятой;
- `g` — выводит число либо в экспоненциальной записи, либо в формате с плавающей запятой, в зависимости от того, как получается короче;
- `o` — выводит восьмеричное представление числа;
- `s` — выводит текстовую строку;

Вот как форматировать выводимые данные с помощью `printf`:

```
awk 'BEGIN{
  x = 100 * 100
  printf "The result is: %e\n", x
}'
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ awk 'BEGIN{
> x = 100 * 100
> printf "The result is: %e\n", x
> }'
The result is: 1.000000e+04
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Форматирование выходных данных с помощью `printf`

Тут, в качестве примера, мы выводим число в экспоненциальной записи. Полагаем, этого достаточно для того, чтобы вы поняли основную идею, на которой построена работа с `printf`.

Задания лабораторной работы

Используя `AWK`:

1. вывести на экран из файла `calendar.txt` день недели и текущее число в виде «сегодня вторник ... августа»;
2. вывести список каталогов, имена которых состоят из русских букв, без дополнительных полей;
3. определить количество(сумму) байтов, занятых всеми вашими текстовыми файлами (`txt`) в каталогах и подкаталогах;
4. определить количество блоков, содержащих ваш текущий каталог;

5. изменить права доступа для некоторых файлов текущего каталога и провести сортировку списка по возможностям доступа;
6. напечатать список каталогов, в которых обнаружены файлы с именами data*.txt;
7. подсчитать, сколько раз пользователь входил в систему;
8. напечатать список пользователей, отсортированный по времени.

Выводы

В данной лабораторной работе мы познакомились с возможностями программируемого фильтра `awk`. Фильтр широко применяется для обработки данных и формирования различного вида отчетов. Для более глубокого изучения всех возможностей фильтра рекомендуется изучить справочные страницы по команде `awk`.