**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Отчет по лабораторной работе № 5

«Программирование на SHELL»

по курсу «ОС Linux»

Грунау Г.Ю.

Студент

Группа АИ-18

Кургасов В.В.

Руководитель

Липецк 2020 г.

# Цель работы

Изучение основных возможностей языка программирования Shell с целью автоматизации процесса администрирования системы за счет написания и использования командных файлов.

Содержание

[Цель работы 2](#_Toc56718187)

[1. Используя команды ECHO, PRINTF вывести информационные сообщения на экран 6](#_Toc56718188)

[2. Присвоить переменной А целочисленное значение. Просмотреть значение переменной А. 6](#_Toc56718189)

[3. Присвоить переменной В значение переменной А. Просмотреть значение переменной В. 6](#_Toc56718190)

[4. Присвоить переменной С значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной. 7](#_Toc56718191)

[5. Присвоить переменной D значение “имя команды”, а именно, команды DATE. Выполнить эту команду, используя значение переменной. 8](#_Toc56718192)

[6. Присвоить переменной E значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной. 8](#_Toc56718193)

[7. Присвоить переменной F значение “имя команды”, а именно сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной. 9](#_Toc56718194)

[8. Программа запрашивает значение переменной, а затем выводит значение этой переменной. 10](#_Toc56718195)

[9. Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной. 10](#_Toc56718196)

[10. Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды a) EXPR; б) ВС)., 12](#_Toc56718197)

[11. Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран. 13](#_Toc56718198)

[12. Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки 14](#_Toc56718199)

[13. Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается. 15](#_Toc56718200)

[14. Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно 16](#_Toc56718201)

[15. Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения. 18](#_Toc56718202)

[16. Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран 19](#_Toc56718203)

[17. Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются 20](#_Toc56718204)

[18. В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc. 22](#_Toc56718205)

[19. Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение 23](#_Toc56718206)

[20. Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла. 24](#_Toc56718207)

[21. Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать а) имена файлов; б) позиционные параметры). 26](#_Toc56718208)

[22. Если файл запуска программы найден, программа запускается (по выбору). 27](#_Toc56718209)

[23. В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране. 28](#_Toc56718210)

[24. Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается. 30](#_Toc56718211)

[25. Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных. 32](#_Toc56718212)

[Вывод 33](#_Toc56718213)

Ход работы

# Используя команды ECHO, PRINTF вывести информационные сообщения на экран

Создадим сценарий, используя редактор vi scrypt.sh и введем код:



Рисунок 1 – Использование ECHO, PRINTF

Присваиваем переменной str значение текущей даты. Затем выведем значение переменной с помощью echo и printf, используя формат строки с переносом “%s\n% и занесем переменную окружения в двойные скобки “$str”, чтобы дата отобразилась одной строкой.

Запустим сценарий:



Рисунок 2 – Вывод даты

# Присвоить переменной А целочисленное значение. Просмотреть значение переменной А.



Рисунок 3 – Присвоение значения



Рисунок 4 – Просмотр значения А

# Присвоить переменной В значение переменной А. Просмотреть значение переменной В.



Рисунок 5 – Присвоение значения В



Рисунок 6 – Просмотр значения В

# Присвоить переменной С значение “путь до своего каталога”. Перейти в этот каталог с использованием переменной.

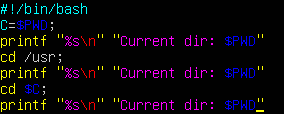


Рисунок 7 – Переход в директорию

C=$PWD;

// Присваиваем переменной текущую директорию

printf “%s\n” “Current dir: $PWD”;

// Выводим текущую директорию

cd /usr;

// Переходим в директорию /usr

printf “%s\n” “Current dir: $PWD”;

// Выводим текущую директорию

cd $C;

// Переходим в директорию, присвоенную переменной С

printf “%s\n” “Current dir: $PWD”;

// Выводим текущую директорию



Рисунок 8 – Результат

На рисунке 8 видно, что нам удалось перейти в начальную директорию с помощью переменной С.

# Присвоить переменной D значение “имя команды”, а именно, команды DATE. Выполнить эту команду, используя значение переменной.

Чтобы узнать текущую дату и время, нужно ввести утилиту date.



Рисунок 9 – Присвоение переменной имени команды

D=`date` // Присвоение команды date переменной D

echo $D // Обращение к переменной D



Рисунок 10 – Выполнение скрипта

Как мы видим на рисунке 10, команда date успешно сработала по обращению к переменной D.

# Присвоить переменной E значение “имя команды”, а именно, команды просмотра содержимого файла, просмотреть содержимое переменной. Выполнить эту команду, используя значение переменной.



Рисунок 11 – Код скрипта

E=`cat < ‘/home/lovediehate/tmpfile’` // Присваивание переменной E значение выполнения команды cat < ‘file’

printf “%s\n” “$E” // Вывод значения переменной E

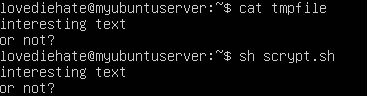


Рисунок 12 – Выполнение скрипта

На рисунке 12 видно, что наш скрипт scrypt.sh успешно вывел содержимое файла tmpfile.

# Присвоить переменной F значение “имя команды”, а именно сортировки содержимого текстового файла. Выполнить эту команду, используя значение переменной.



Рисунок 13 – Код скрипта

D=`sort ‘/home/lovediehate/tmpfile’`; // Присвоение переменной D результата команды /home/lovediehate/tmpfile.

printf “%s\n” “$D” // Вывод значения переменной D

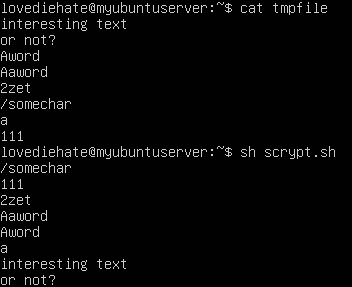


Рисунок 14 – Выполнение скрипта

Результат выполнение скрипта: строки файла tmpfile успешно выведены и предварительно отсортированы.

# Программа запрашивает значение переменной, а затем выводит значение этой переменной.

За пользовательский ввод в BASH-скриптах отвечает встроенная команда read, которая считывает одну строку в переменную.



Рисунок 15 – Код скрипта

Строка read F подразумевает, что пользователь должен ввести в консоли строковое значение, которое присвоится переменной F.



Рисунок 16 – Выполнение скрипта

Видно, что пользователь (я) после консольного сообщения “Enter string:” ввёл какую-то строку, она считалась в переменную и после этого вывелась.

# Программа запрашивает имя пользователя, затем здоровается с ним, используя значение введенной переменной.



Рисунок 17 – Код скрипта

Строка read Name подразумевает, что пользователь должен ввести в консоли строковое значение (имя), которое присвоится переменной Name.

В последней строке скрипт выводит сообщение Hello, $Name. Вместо $Name выведется строка, которую ввёл пользователь (либо пустота, если пользователь ничего не ввёл).

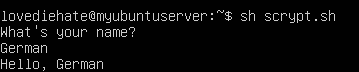


Рисунок 18 – Выполнение скрипта

# Программа запрашивает значения двух переменных, вычисляет сумму (разность, произведение, деление) этих переменных. Результат выводится на экран (использовать команды a) EXPR; б) ВС).,

Для вычисления выражений можно использовать команду expr. Её можно применять как в командной строке, так и в скриптах.

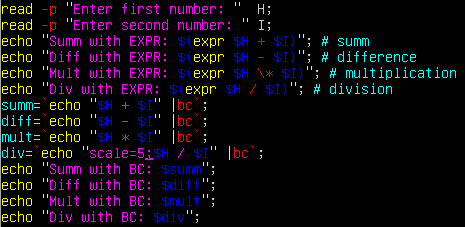


Рисунок 19 – Код скрипта

С помощью expr доступно только целочисленное вычисление. Для дробных чисел лучше использовать bc – Си-подобный интерактивный интерпретатор. Для того, чтобы выводились числа после запятой, нужно использовать переменную scale, означающую количество цифр в дробной части (10 строка, рис. 19).

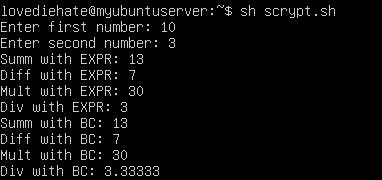


Рисунок 20 – Выполнение скрипта

Как мы видим, при делении 10 на 3 EXPR выдал результат 3, а BC выдал результат 3.33333. Всё благодаря тому, что при делении с BC была указана переменная scale=5.

# Вычислить объем цилиндра. Исходные данные запрашиваются программой. Результат выводится на экран.



Рисунок 21 – Код программы

read –p “Enter the height: “ h; // считывание введённой строки в h

Параметр –p <PROMT> - строка приглашения <PROMPT>. Без оконечного перевода строки, обычно в ней выводят подсказку перед тем, как команда read будет считывать данные.

Объем цилиндра вычисляется по формуле V=3.14\*r\*r\*h, где r – радиус, h - высота;

Т.к. при вычислении объема цилиндра используется число Пи, нужно использовать калькулятор bc, т.к. EXPR работает только с целыми числами.

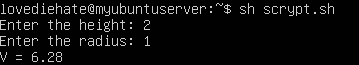


Рисунок 22 – Выполнение программы

# Используя позиционные параметры, отобразить имя программы, количество аргументов командной строки, значение каждого аргумента командной строки

Позиционные параметры — это аргументы командой строки (или функции в скрипте), доступ к которые осуществляется по номеру ${number}.

Специальные параметры:

$\* — все аргументы;

$@ — все аргументы;

$# — количество аргументов;

$0 — имя скрипта;

$$ — PID процесса;

$! — PID последнего процесса в background-е;

$? — результат выполнения выражения или скрипта (0 — если удачно, 1 — если ошибка);

$\_ — последний аргумент.

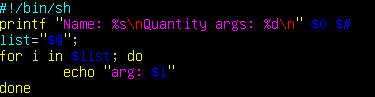


Рисунок 23 – Код скрипта

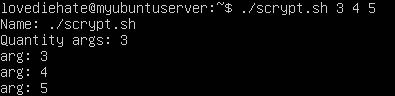


Рисунок 24 – Выполнение скрипта

С помощью $0 вывелось имя, $# – кол-во аргументов, и с помощью цикла и обращения к позиционным параметрам вывелись все введённые аргументы.

# Используя позиционный параметр, отобразить содержимое текстового файла, указанного в качестве аргумента командной строки. После паузы экран очищается.

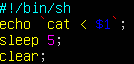


Рисунок 25 – Код скрипта

Скрипт будет выводить содержание файла (путь к файлу передаётся в первом позиционном аргументе), затем после пятисекундной паузы очистит терминал.

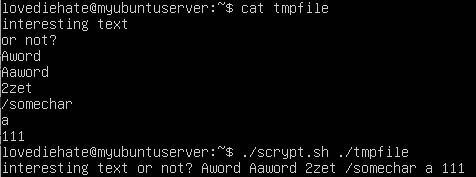


Рисунок 26 – Выполнение скрипта

На рисунке 26 изображены просмотр файла через cat и выполнение скрипта.

# Используя оператор FOR, отобразить содержимое текстовых файлов текущего каталога поэкранно

Для поэкранного отображения используется команда less. С её помощью можно посмотреть содержимое нескольких файлов по очереди, при этом окно терминала не будет засоряться.

Чтобы получить список файлов каталога, используется простая команда ls. Цикл пробегается по списку файлов и определяет формат каждого файла с помощью утилиты file с параметром –b

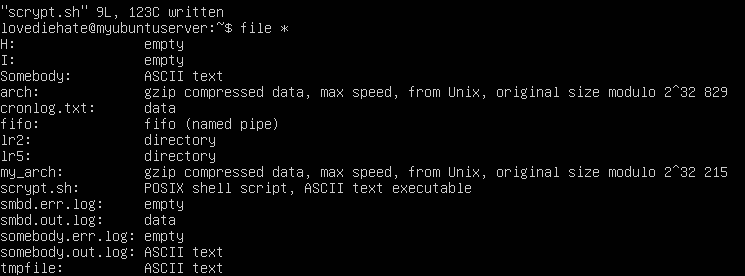


Рисунок 27 – Демонстрация утилиты file

Параметр -b убирает из вывода имя файла, оставляя только его тип. Это требуется для более простого дальнейшего сравнения. Далее, формат каждого элемента списка с помощью условного оператора if проверяется, и если формат списка – “ASCII text”, то выполняется просмотр содержимого этого файла с помощью команды less “путь/имя файла”. $PWD – путь к текущему каталогу, $i – имя файла из списка файлов ls. На рисунке 27 видно, что у нас всего 3 текстовых файла. Их содержимое скрипт и должен вывести.

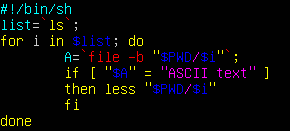


Рисунок 28 – Код программы



Рисунок 29 – Выполнение скрипта, первый файл

Скрипт нашёл в каталоге текстовый файл и отобразил его содержимое. Далее, чтобы перейти к следующему файлу, нужно ввести :q (так как less применяется отдельно к каждому файлу, а не списку файлов, для просмотра следующего файла нужно выйти из текущего less, тогда итератор цикла перейдёт к следующему файлу списка и снова применит к нему less. 

Рисунок 30 – Выполнение скрипта, второй файл



Рисунок 31 – Выполнение скрипта, третий файл

# Программой запрашивается ввод числа, значение которого затем сравнивается с допустимым значением. В результате этого сравнения на экран выдаются соответствующие сообщения.

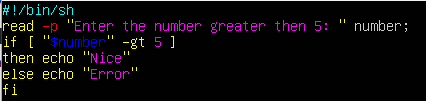


Рисунок 32 – Код скрипта

Скрипт предложит пользователю ввести число. Если оно будет больше 5, то скрипт выдаст сообщение ‘Nice’, иначе ‘Error’.

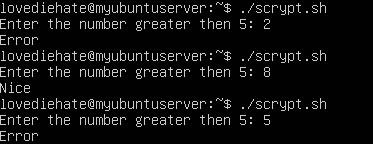


Рисунок 33 – Выполнение скрипта

# Программой запрашивается год, определяется, високосный ли он. Результат выдается на экран

Високосный год – год, который делится на 4 без остатка. Скрипт делит по модулю входные данные на 4, и если остатка равен нулю, значит год високосный (Leap), иначе не високосный (Not leap).

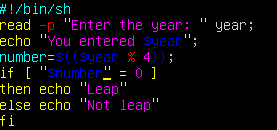


Рисунок 34 – Код программы

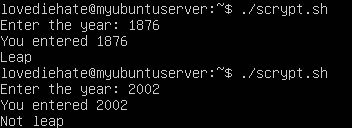


Рисунок 35 – Выполнение программы

# Вводятся целочисленные значения двух переменных. Вводится диапазон данных. Пока значения переменных находятся в указанном диапазоне, их значения инкрементируются

for ((;;)) do – бесконечный цикл.

Далее идут 3 ветки условий if.

Первая ветка – проверка введённых А и В на вход в диапазон.

Вторая ветка – увеличивает А на 1 каждый проход цикла, если А меньше второй границы диапазона.

Третья ветка – такая же, как вторая, только для переменной B.

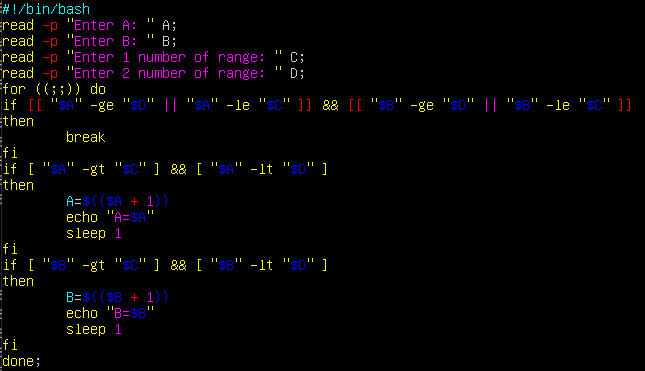


Рисунок 36 – Код программы

В первом случае только А входит в диапазон от [1;3], поэтому переменная В не изменилась, а переменная А инкрементировалась (сработало второе условие).

Во втором случае обе переменные вошли в диапазон и увеличились до правой границы диапазона (сработало второе и третье условие).

В третьем случае ни одна переменная не вошла в диапазон, поэтому они не изменились (сработало первое условие – break из цикла).

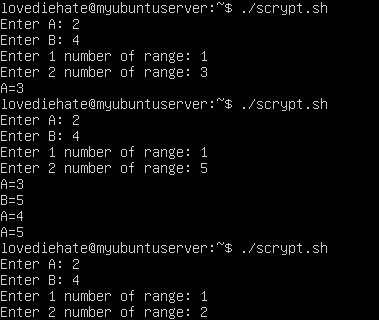


Рисунок 37 – Выполнение скрипта

# В качестве аргумента командной строки указывается пароль. Если пароль введен верно, постранично отображается в длинном формате с указанием скрытых файлов содержимое каталога /etc.

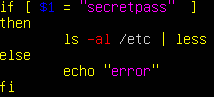


Рисунок 38 – Код скрипта



Рисунок 39 –Ввод неправильного пароля

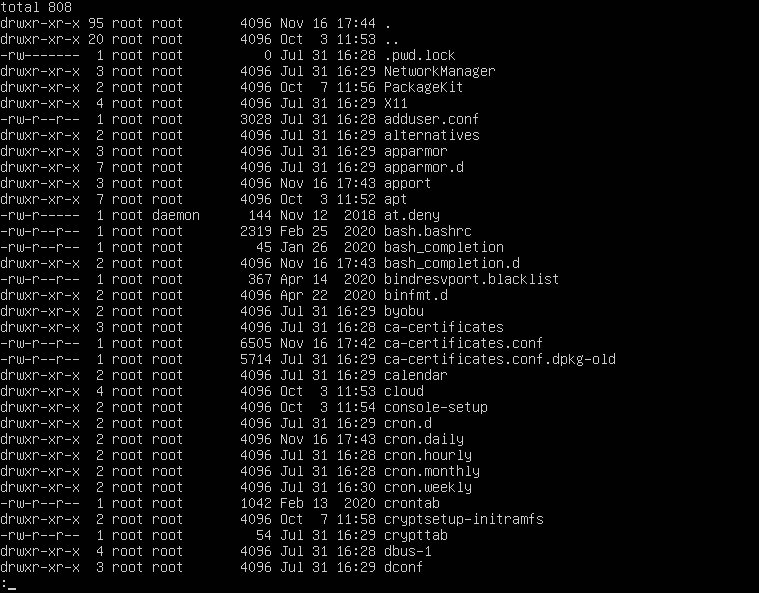


Рисунок 40 – Ввод правильного пароля

# Проверить, существует ли файл. Если да, выводится на экран его содержимое, если нет - выдается соответствующее сообщение

Для проверки существования файла применяется параметр –s.

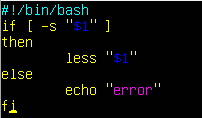


Рисунок 41 – Код скрипта

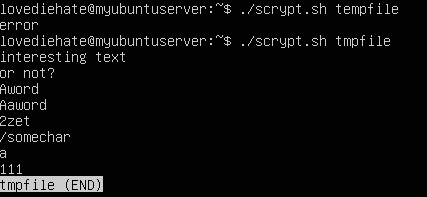


Рисунок 42 – Выполнение скрипта

Файла tempfile не существует. Файл tmpfile существует, содержимое вывелось.

# Если файл есть каталог и этот каталог можно читать, просматривается содержимое этого каталога. Если каталог отсутствует, он создается. Если файл не есть каталог, просматривается содержимое файла.

В самом первом условии файл проверяется на существование. Если не существует, то создастся c помощью mkdir. Если существует, то будут проверяться вложенные условия.

В первом вложенном условии файл проверяется на то, является ли он каталогом и доступен ли для чтения. Если да, то выводится список его файлов с помощью ls.

Во втором вложенном условии файл проверяется на то, является ли он обычным файлом. Если да, то выводится его содержимое с помощью less.

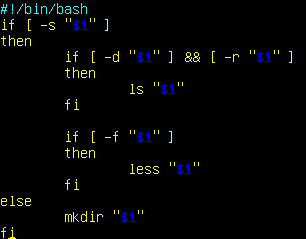


Рисунок 43 – Код скрипта

Синим цветом выделяются названия каталогов.

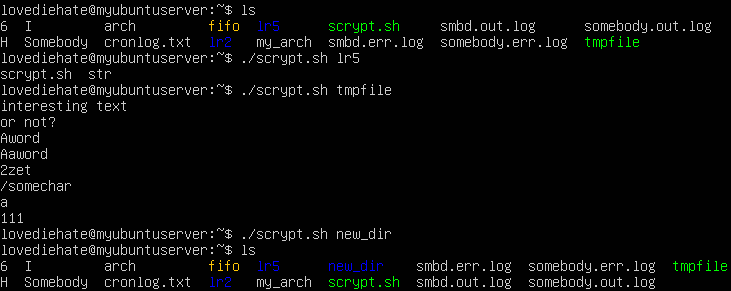


Рисунок 44 – Выполнение программы

В первом случае вводится существующая директория lr5. Скрипт выдал содержимое, т.е. список файлов этой директории.

Во втором случае вводится существующий файл tmpfile. Скрипт отображает его содержимое.

В третьем случае вводится несуществующий файл. Скрипт создал каталог с таким названием.

# Анализируются атрибуты файла. Если первый файл существует и используется для чтения, а второй файл существует и используется для записи, то содержимое первого файла перенаправляется во второй файл. В случае несовпадений указанных атрибутов или отсутствия файлов на экран выдаются соответствующие сообщения (использовать а) имена файлов; б) позиционные параметры).

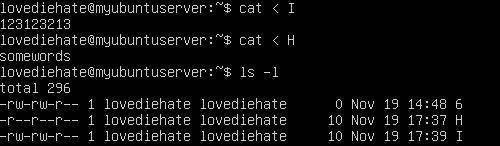


Рисунок 45 – Просмотр содержимого файлов и доступа

Сначала первый файл проверяется на существование и на чтение -r. Если он прошёл проверку, то проверяется второй файл на запись -w. Если проходит проверку, то содержимое первого записывается в конец второго файла.

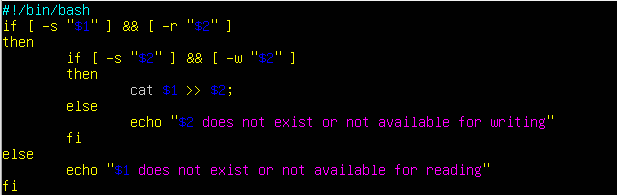


Рисунок 46 – Код программы

Файл H доступен только для чтения, значит в него нельзя ничего записать. Файл I доступен для записи, значит в него можно записать.

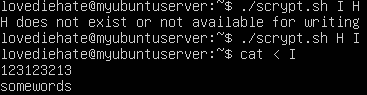


Рисунок 47 – Выполнение программы

Попытка записать из файла I в файл H не увенчалась успехом.

# Если файл запуска программы найден, программа запускается (по выбору).



Рисунок 48 – Код скрипта

Весь код скрипта:

$1



Рисунок 49 – echo $PATH

Программы из перечня директорий, указанных в PATH, могут запускаться без пути к ним. Система ищет в этих папках введённую программу, и если находит, то запускает. Если нет, то выводит ошибку command not found.



Рисунок 50 – Выполнение скрипта

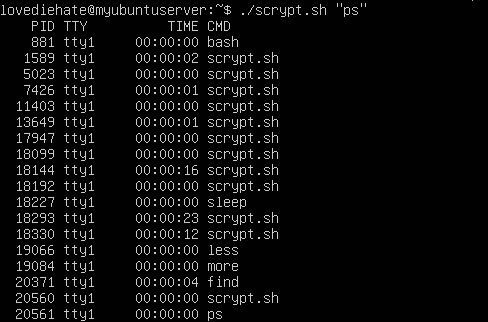


Рисунок 51 – Выполнение скрипта

# В качестве позиционного параметра задается файл, анализируется его размер. Если размер файла больше нуля, содержимое файла сортируется по первому столбцу по возрастанию, отсортированная информация помещается в другой файл, содержимое которого затем отображается на экране.

Вес файла можно узнать с помощью команды wc –c filename.

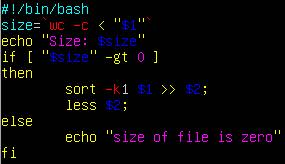


Рисунок 52 – Код скрипта



Рисунок 53 – Выполнение скрипта

Первый файл пуст, поэтому его нельзя отсортировать.



Рисунок 54 – Выполнение скрипта

Файл sortedfile имеет вес 22, поэтому он ввёлся в отсортированном виде в файл new\_empty\_file. Содержимое второго файла отображено.

# Командой TAR осуществляется сборка всех текстовых файлов текущего каталога в один архивный файл my.tar, после паузы просматривается содержимое файла my.tar, затем командой GZIP архивный файл my.tar сжимается.

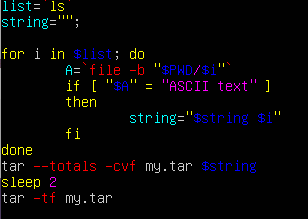


Рисунок 55 – Код скрипта

Цикл проходит по списку всех файлов и создаёт список текстовых файлов посредством конкатенации string=”$string $i”. Затем этот список используется для архивирования.

tar –totals –cvf my.tar $string // Архивирование с выводом итоговой информации завершенного процесса (--totals), с созданием архива my.tar (–c), выводом подробной информации процесса (-v) и выводом результата в файл (-f).

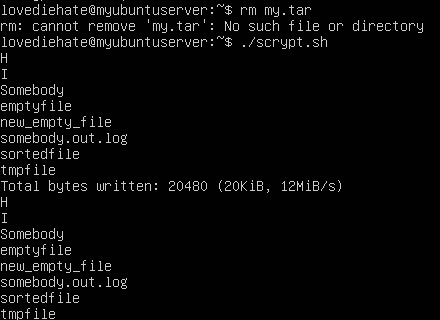


Рисунок 56 – Выполнение программы

На рисунке 56 видно, что архива не было до запуска скрипта. Он успешно создался в скрипте и заполнился текстовыми файлами.

# Написать скрипт с использованием функции, например, функции, суммирующей значения двух переменных.

Синтаксис создания функции очень прост:

имя\_функции() { список\_команд }

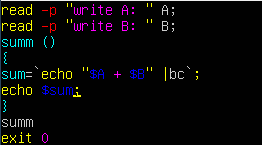


Рисунок 57 – Код

В функции sum выполняется сложение двух введённых переменных $A+$B с помощью калькулятора bc и и вывод результата.

В предпоследней строке (рис. 57) осуществляется вызов функции summ, а в последней осуществляется завершение работы сценария exit 0.



Рисунок 58 – Выполнение скрипта

# Вывод

Я изучил основные возможности языка программирования Shell с целью автоматизации процесса администрирования системы засчёт написания и использования командных файлов.