**Практическая работа №1**

**По дисциплине «Системное программное обеспечение»**

**На тему «Использование командной строки Linux»**

**Цель работы:**

Получить навыки по работе с командной строкой операционной системы Linux при помощи дистрибутива Debian 10.

**Часть 1. Основные команды командной строки**

1. Используя VirtualBox создать виртуальную машину с Debian 10;
2. Познакомиться с базовыми командами работы с файловой системой:

* cd – команда для смены директории, работа с абсолютными и относительными путями;
* pwd – команда отображения текущего каталога;
* ls – команда просмотра содержимого директории, разобраться с флагами –a, -l;
* mkdir – команда создания директорий;
* touch – команда созданий файла;
* echo – команда вывода данных в консоль, перенаправление потока вывода в файл при помощи > и >>;
* cat – команда вывода данных из файла;
  + Вывод текста из файла

cat filename

* + Ввод текста в файл

cat > test

text

test

123

Ctrl + D

* + Ввод многострочного текста в файл

cat << EOF >> test

test123

EOF

* rm и rmdir – команды удаления файлов и каталогов;
* mv – команда перемещения файлов и каталогов;
* cp – команда копирования файлов и каталогов;
* find – поиск файлов;
* more, less, head, tail для вывода информации из файла.

1. Познакомиться с командами управления правами

* Команда chmod

**chmod [опции] <права> </путь/к/файлу>**

Есть три основных вида прав:

**r - чтение;**

**w - запись;**

**x - выполнение;**

**s - выполнение от имени суперпользователя (дополнительный);**

Также есть три категории пользователей, для которых вы можете установить эти права на файл linux:

**u - владелец файла;**

**g - группа файла;**

**o - все остальные пользователи.**

Синтаксис настройки прав такой:

группа\_пользователейдействиевид\_прав

В качестве действий могут использоваться знаки "+" - включить или "-" - отключить. Рассмотрим несколько примеров:

u+x - разрешить выполнение для владельца;

ugo+x - разрешить выполнение для всех;

ug+w - разрешить запись для владельца и группы;

o-x - запретить выполнение для остальных пользователей;

ugo+rwx - разрешить все для всех;

Но права можно записывать не только таким способом. Есть еще восьмеричный формат записи, он более сложен для понимания, но пишется короче и проще.

0 - никаких прав;

1 - только выполнение;

2 - только запись;

3 - выполнение и запись;

4 - только чтение;

5 - чтение и выполнение;

6 - чтение и запись;

7 - чтение запись и выполнение.

Права на папку linux такие же, как и для файла. Во время установки прав сначала укажите цифру прав для владельца, затем для группы, а потом для остальных. Например:

744 - разрешить все для владельца, а остальным только чтение;

755 - все для владельца, остальным только чтение и выполнение;

764 - все для владельца, чтение и запись для группы, и только чтение для остальных;

777 - всем разрешено все.

Каждая из цифр не зависит от предыдущих, вы вбираете именно то, что вам нужно. Теперь давайте рассмотрим несколько опций команды, которые нам понадобятся во время работы:

-c - выводить информацию обо всех изменениях;

-f - не выводить сообщения об ошибках;

-v - выводить максимум информации;

--preserve-root - не выполнять рекурсивные операции для корня "/";

--reference - взять маску прав из указанного файла;

-R - включить поддержку рекурсии;

--version - вывести версию утилиты.

* Команда chown

**chown <пользователь> [опции] </путь/к/файлу>**

В поле пользователь надо указать пользователя, которому мы хотим передать файл. Также можно указать через двоеточие группу, например, **пользователь:группа**. Тогда изменится не только пользователь, но и группа. Вот основные опции, которые могут вам понадобиться:

-c, --changes - подробный вывод всех выполняемых изменений;

-f, --silent, --quiet - минимум информации, скрыть сообщения об ошибках;

--dereference - изменять права для файла к которому ведет символическая ссылка вместо самой ссылки (поведение по умолчанию);

-h, --no-dereference - изменять права символических ссылок и не трогать файлы, к которым они ведут;

--from - изменять пользователя только для тех файлов, владельцем которых является указанный пользователь и группа;

-R, --recursive - рекурсивная обработка всех подкаталогов;

-H - если передана символическая ссылка на директорию - перейти по ней;

-L - переходить по всем символическим ссылкам на директории;

-P - не переходить по символическим ссылкам на директории (по умолчанию).

1. Перенаправление потока вывода

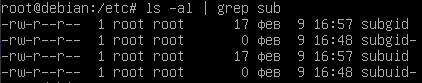
Как было сказано выше, существует возможность перенаправить вывод команды из консоли в файл при помощи следующих команд:

* > - перезапись файла;
* >> - добавление в файл.

Существует также вариант перенаправления вывода на ввод другой команде.

**| -** данный символ как раз позволяет это сделать.

Пример:



Команда **grep** позволяет выводить только те строки, в которых было найдено совпадение с заданным шаблоном. В данном случае при помощи команды **ls -al** был получен список всех файлов из директории /etc, который затем был отфильтрован по присутствию в них строки **sub**.

**Часть 2. Анализ текста**

Данная часть практической работы посвящена разбору и анализу текста при помощи командной строки linux.

Для этого возможно использовать команды **sed** и **awk** а также регулярные выражения.

Предлагается ознакомиться с данными темами по следующим ссылкам:

**sed**: <https://habr.com/ru/company/ruvds/blog/327530/>

**awk**: <https://habr.com/ru/company/ruvds/blog/327754/>

**регулярные выражения**: <https://habr.com/ru/company/ruvds/blog/327896/>

После ознакомлениями необходимо выполнить следующие задания:

* Удалить n-ую строку.
* Удалить все пустые строки.
* Удалить все строки до первой пустой строки, включительно.
* Вывести строки, содержащие определенное слово.
* В каждой строке, заменить первое встретившееся слово "Windows" на слово "Linux".
* В каждой строке, заменить одно слово на другое по всему тексту.
* Удалить все пробелы в конце каждой строки.
* Заменить все последовательности ведущих нулей одним символом "0".
* Удалить все строки, содержащие определенное слово.
* Удалить все найденные вхождения выбранного слова, оставляя остальную часть строки без изменений.

**Часть 3. Bash-скрипты**

Данная часть работы посвящена написанию bash-скриптов.

По сути своей Bash-скрипты представляют из себя ни что иное как последовательность команд командной строки, объединенных в один файл для решения какой-либо задачи

Любой скрипт начитается с так называемой шебанг строки, которая указывает на расположение исполняемого файла той оболочки, для которой мы хотим написать скрипт. В нашем случае это bash, следовательно данная строка будет выглядеть следующим образом:

**#!/bin/bash**

После чего уже следует непосредственно сам скрипт.

Символом решетки в bash-скриптах обозначаются комментарии.

Напишем самый базовый скрипт, который просто будет выводить текущий каталог. Создадим файл **myscript** и запишем в него следующий код.

#!/bin/bash

*# This is a comment*

pwd

Запуск скриптов из командной строки осуществляется следующим образом. В директории с файлом скрипта необходимо написать следующую команду **./myscript**.

Однако просто так этого сделать не получится, система выдаст ошибку, необходимо выдать этому файлу права на выполнение.

chmod +x ./myscript

После чего скрипт может быть запущен.

Для вывода текста в скриптах можно воспользоваться уже известной командой **echo**.

**Переменные**

Как и любой другой язык bash обладает возможностью создания переменных.

Переменные бывают как пользовательскими, создаваемыми при работе скрипта, так и переменными среды, созданными для хранения параметров ОС.

Для обращения к переменным среды следует использовать следующую конструкцию: **$ИМЯ\_ПЕРЕМЕННОЙ.**

К примеру, выведем в сообщении путь к домашней директории текущего пользователя.

#!/bin/bash

*# display user home*

echo "Home for the current user is: $HOME"

Заметьте, что строковый литерал – двойные кавычки – никак не повлиял на распознавание переменной среды.

Пользовательские переменные в bash-скриптах имеют динамический тип. Вот пример их задания с последующим выводом.

#!/bin/bash

*# testing variables*

grade=5

person="Adam"

echo "$person is a good boy, he is in grade $grade"

В данном случае было создано 2 переменные: **grade** и **person**.

**Подстановка команд**

Одна из самых полезных возможностей bash-скриптов — это возможность извлекать информацию из вывода команд и назначать её переменным, что позволяет использовать эту информацию где угодно в файле сценария.

Сделать это можно двумя способами.

* С помощью значка обратного апострофа «`»
* С помощью конструкции $()

Используя первый подход, проследите за тем, чтобы вместо обратного апострофа не ввести одиночную кавычку. Команду нужно заключить в два таких значка:

mydir=`pwd`

При втором подходе то же самое записывают так:

mydir=$(pwd)

А скрипт, в итоге, может выглядеть так:

#!/bin/bash

mydir=$(pwd)

echo $mydir

В ходе его работы вывод команды pwd будет сохранён в переменной mydir, содержимое которой, с помощью команды echo, попадёт в консоль.

**Математические операции**

Для выполнения математических операций в файле скрипта можно использовать конструкцию вида **$(( a + b ))**:

#!/bin/bash

var1=$(( 5 + 5 ))

echo $var1

var2=$(( $var1 \* 2 ))

echo $var2

var3=$(( $var2 / 4 ))

echo $var3

var4=$(( $var3 % 5 ))

echo $var4

var5=$(( $var4 \*\* 2 ))

echo $var5

**Управляющая конструкция if-then-else**

Для управления потоком исполнения команд в bash-скриптах можно использовать управляющую конструкцию if-then-else. Работает она как во всех языках программирования, однако имеет свои особенности. Вот её подробный синтаксис:

if команда1

then

команды

elif команда2

then

команды

else

команды

fi

В отличии от большинства известных вам на данный момент языков bash-скрипты обладают возможностью задать дополнительные условия в конструкции **if-else** при помощи ключевого слова **elif**. Пример скрипта с **if-then-else**.

#!/bin/bash

user=anotherUser

if grep $user /etc/passwd

then

echo "The user $user Exists"

else

echo "The user $user doesn’t exist"

fi

Пример скрипта с **elif**.

#!/bin/bash

user=anotherUser

if grep $user /etc/passwd

then

echo "The user $user Exists"

elif ls /home

then echo "The user doesn’t exist but anyway there is a directory under /home"

fi

Команда grep используется для поиска информации о пользователе, чье имя записано в переменной **user**, в файле /etc/passwd, который содержит информацию обо всех пользователях в системе.

**Сравнение чисел**

В отличии от привычных операций сравнения в языках программирования bash-скрипты обладают специфическим синтаксисом операций сравнения. Например, дл  
я чисел применяются следующие операции:

* n1 -eq n2 – Возвращает истинное значение, если n1 равно n2.
* n1 -ge n2 – Возвращает истинное значение, если n1больше или равно n2.
* n1 -gt n2 – Возвращает истинное значение, если n1 больше n2.
* n1 -le n2 – Возвращает истинное значение, если n1меньше или равно n2.
* n1 -lt n2 – Возвращает истинное значение, если n1 меньше n2.
* n1 -ne n2 – Возвращает истинное значение, если n1не равно n2.

При использовании подобных операций выражения необходимо заключать в квадратные скобки, например:

#!/bin/bash

val1=6

if [ $val1 -gt 5 ]

then

echo "The test value $val1 is greater than 5"

else

echo "The test value $val1 is not greater than 5"

fi

**Сравнение строк**

Для сравнения строк используются чуть более привычные операторы, однако и они не лишены новшеств.

* str1 = str2 – Проверяет строки на равенство, возвращает истину, если строки идентичны.
* str1 != str2 – Возвращает истину, если строки не идентичны.
* str1 < str2 – Возвращает истину, если str1меньше, чем str2.
* str1 > str2 – Возвращает истину, если str1больше, чем str2.
* -n str1 – Возвращает истину, если длина str1больше нуля.
* -z str1 – Возвращает истину, если длина str1равна нулю.

Пример сравнения строк

#!/bin/bash

user ="likegeeks"

if [$user = $USER]

then

echo "The user $user  is the current logged in user"

fi

При работе с операторами < и > есть несколько особенностей, которые стоит учитывать. Во-первых, их необходимо экранировать при помощи символа \. Во-вторых, необходимо заключать имена переменных в двойные кавычки "$val2". Приведем пример работы с подобным оператором.

#!/bin/bash

val1=text

val2="another text"

if [ "$val1" \> "$val2" ]

then

echo "$val1 is greater than $val2"

else

echo "$val1 is less than $val2"

fi

**Проверки файлов**

Наиболее используемыми командами для bash скриптов являются команды проверки файлов.

-d file – Проверяет, существует ли файл, и является ли он директорией.

-e file – Проверяет, существует ли файл.

-f file – Проверяет, существует ли файл, и является ли он файлом.

-r file – Проверяет, существует ли файл, и доступен ли он для чтения.

-s file – Проверяет, существует ли файл, и не является ли он пустым.

-w file – Проверяет, существует ли файл, и доступен ли он для записи.

-x file – Проверяет, существует ли файл, и является ли он исполняемым.

file1 -nt file2 – Проверяет, новее ли file1, чем file2.

file1 -ot file2 – Проверяет, старше ли file1, чем file2.

-O file – Проверяет, существует ли файл, и является ли его владельцем текущий пользователь.

-G file – Проверяет, существует ли файл, и соответствует ли его идентификатор группы идентификатору группы текущего пользователя.

Пример скрипта, в котором используются эти команды.

#!/bin/bash

mydir=/home/likegeeks

if [ -d $mydir ]

then

echo "The $mydir directory exists"

cd $mydir

ls

else

echo "The $mydir directory does not exist"

fi

**Операторы цикла**

Bash-скрипты поддерживают несколько вариантов циклов для перебора последовательностей значений. В число этих циклов входят:

* for;
* while.

Для начала рассмотрим цикл **for**. Базовая структура такого цикла выглядит следующим образом.

for var in list

do

команды

done

Самый простой пример – перебор списка простых значений.

#!/bin/bash

for var in first second third fourth fifth

do

echo The $var item

done

Для перебора сложных значений необходимо заключать это значения в строковые литералы – двойные кавычки, к примеру:

#!/bin/bash

for var in first "the second" "the third" "I’ll do it"

do

echo "This is: $var"

done

Список для цикла for может быть получен из результата выполнения какой-либо команды, полученного при помощи рассмотренной ранее подстановки команд. Как пример – перебор вывода команды cat.

#!/bin/bash

file="myfile"

for var in $(cat $file)

do

echo " $var"

done

Однако проблема такого вывода в том, что файл будет обрабатываться по словам, а не по строкам, как того хотелось бы.

Это связано с тем, что оболочка bash считает разделителем строки следующий набор символов:

* Пробел;
* Знак табуляции;
* Знак перевода строки.

Для того, чтобы задать разделитель необходимо использовать переменную окружения IFS (Internal Field Separator). Пример того, как можно выполнить итерацию лишь по строкам файла.

#!/bin/bash

file="/etc/passwd"

IFS=$'\n'

for var in $(cat $file)

do

echo " $var"

done

По мере работы скрипта возможно изменение переменной IFS, так что она может задаваться в зависимости от контекста.

Для обхода файлов, находящихся в директориях, также используется цикл for. Вот пример такого обхода с выводом списка файлов и директорий.

#!/bin/bash

for file in /home/<username>/\*

do

if [ -d "$file" ]

then

echo "$file is a directory"

elif [ -f "$file" ]

then

echo "$file is a file"

fi

done

Как можно видеть из примера для обхода файлов можно использовать следующий путь, который подразумевает сбор всех файлов и директорий - **/home/<username>/\***

Вторая разновидность циклов – цикл while.

while команда проверки условия

do

другие команды

done

Пример скрипта с таким циклом.

#!/bin/bash

var1=5

while [ $var1 -gt 0 ]

do

echo $var1 var1=$[ $var1 - 1 ]

done

Для управления циклами также могут применяться привычные команды операторы **break** и **continue**.

Результат работы цикла также могут быть выведены в файл при помощи перенаправления вывода. Вот пример такого скрипта.

#!/bin/bash

for (( a = 1; a < 10; a++ ))

do

echo "Number is $a"

done > myfile.txt

echo "finished."

Более подробно с написанием скриптов можно ознакомиться по следующим ссылкам:

* <https://habrahabr.ru/company/ruvds/blog/325522/>
* <https://habr.com/ru/company/ruvds/blog/325928/>

Варианты для выполнения.

Данные хранятся в файле MOCK\_DATA.csv. Вывод результатов необходимо производить в файл. Вывод должен содержать номер строки, а также найденное значение.

|  |  |
| --- | --- |
| **№** | **Задание** |
| 1 | Найти все строки, в которых ip имеет в третьем октете значение 31 |
| 2 | Найти все строки, в которых URL содержит в себе домен uk |
| 3 | Найти все строки, в которых содержатся файлы с расширением mov |
| 4 | Найти все строки, в которых email адрес имеет домен amazon |
| 5 | Найти все строки, в которых ip имеет во втором октете значение 141 |
| 6 | Найти все строки, в которых URL содержит в себе домен edu |
| 7 | Найти все строки, в которых содержатся файлы с расширением txt |
| 8 | Найти все строки, в которых email адрес имеет домен yahoo |
| 9 | Найти все строки, в которых ip имеет в четвертом октете значение 95 |
| 10 | Найти все строки, в которых URL ссылается на файл с уровнем вложенности 3 (пример: https://wiki.com/assets/images/image.jpg) |
| 11 | Найти все строки, в которых содержатся файлы с расширением mpeg |
| 12 | Найти все строки, в которых email адрес содержит число вида 9X |
| 13 | Найти все строки, в которых ip имеет в первом октете значение 46 |
| 14 | Найти все строки, в которых URL ссылается на файл с уровнем вложенности 4 (пример: https://wiki.com/assets/images/jpg/image.jpg) |
| 15 | Найти все строки, в которых содержатся файлы с расширением gif |
| 16 | Найти все строки, в которых email адрес содержит число 0 |

**Отчет**

В результате выполнения всех заданий в отчет необходимо включить следующее:

1. Титульный лист;
2. Задание;
3. Выполнение заданий;
   1. Часть 1. Скриншоты выполнения команд с небольшим описанием выполнения.
   2. Часть 2. Скриншоты выполненных заданий, а также команды для их выполнения.
   3. Часть 3. Результат выполнения bash-скрипта, а также текст самого скрипта.
4. Небольшой вывод по проделанной работе.