Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Е.В. Красавин, Е.А. Черепков

ПРОГРАММИРОВАНИЕ В SHELL

Методические указания к домашней работе

по дисциплине «Операционные системы»

Калуга – 2019

УДК 004.62
ББК 32.972.1
К78

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационные технологии».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/04 от 20 ноября 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 5 от «25» ноября 2019 г.

Председатель методической комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им. Н.Э. Баумана протокол № 3 от «3» 12 2019 г.

Председатель методической комиссии КФ МГТУ им. Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ _____ А.Б. Лачихина

Авторы
к.т.н., доцент кафедры ИУ4-КФ _____ Е.В. Красавин
ассистент кафедры ИУ4-КФ _____ Е.А. Черепков

Аннотация

Методические указания к домашней работе по курсу «Операционные системы» включают краткие теоретические сведения о shell-сценариях. Содержат описание принципов работы, написания и запуска скриптов.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой дисциплины «Операционные системы» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания, ориентированные на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержат основные сведения о написании Shell-сценариев для операционной системы Linux.

Методические указания составлены для ознакомления студентов с основополагающими понятиями и принципами написании Shell-сценариев для администрирования системы. Для выполнения домашней работы студенту необходимы минимальные навыки программирования.

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью выполнения домашней работы является получение практических навыков по написанию Shell-сценариев для ОС Linux.

Основными задачами выполнения домашней работы являются:

1. Самостоятельно изучить синтаксис и важнейшие структуры Shell-сценариев.
2. Научиться применять, Shell-сценарии для администрирования системы.
3. Закрепить полученные в ходе выполнения лабораторных работ навыки.

Результатами работы являются:

1. Исполняемый файл, содержащий Shell-сценарий, разработанный согласно варианту;
2. Подготовленный отчет.

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Shell — это командная оболочка. Но это не просто промежуточное звено между пользователем и операционной системой, это еще и мощный язык программирования. Программы на языке Shell называют сценариями, или скриптами. Фактически, из скриптов доступен полный набор команд, утилит и программ UNIX. Если этого недостаточно, то к существуют внутренние команды Shell - условные операторы, операторы циклов и пр., которые увеличивают мощь и гибкость сценариев. Shell-скрипты исключительно хороши при программировании задач администрирования системы и др., которые не требуют для своего создания полновесных языков программирования.

Знание языка командной оболочки является залогом успешного решения задач администрирования системы. Во время загрузки Linux выполняется целый ряд сценариев из /etc/rc.d, которые настраивают конфигурацию операционной системы и запускают различные сервисы, поэтому очень важно четко понимать эти скрипты и иметь достаточно знаний, чтобы вносить в них какие-либо изменения.

Язык сценариев легок в изучении, в нем не так много специфических операторов и конструкций. Синтаксис языка достаточно прост и прямолинеен, он очень напоминает команды, которые приходится вводить в командной строке. Короткие скрипты практически не нуждаются в отладке, и даже отладка больших скриптов отнимает весьма незначительное время.

Shell-скрипты очень хорошо подходят для быстрого создания прототипов сложных приложений, даже не смотря на ограниченный набор языковых конструкций и определенную "медлительность". Такой метод позволяет детально проработать структуру будущего приложения, обнаружить возможные "ловушки" и лишь затем приступить к написанию кода на C, C++, Java, или Perl.

Скрипты возвращают нас к классической философии UNIX - "разделяй и властвуй" т.е. разделение сложного проекта на ряд простых

подзадач. Такой подход считается наилучшим или, по меньшей мере, наиболее эстетичным способом решения возникающих проблем, нежели использование нового поколения языков - "все-в-одном", таких как Perl.

Название BASH — это аббревиатура от "Bourne-Again Shell" и игра слов от, ставшего уже классикой, "Bourne Shell" Стефена Бурна (Stephen Bourne). В последние годы BASH достиг такой популярности, что стал стандартной командной оболочкой для многих разновидностей UNIX. Большинство принципов программирования на BASH одинаково хорошо применимы и в других командных оболочках, таких как Korn Shell (ksh), от которой Bash позаимствовал некоторые особенности, а также C Shell и его производные.

СИНТАКСИС SHELL-СЦЕНАРИЕВ

Служебные символы, используемые в текстах сценариев.

Комментарии. Строки, начинающиеся с символа # (за исключением комбинации #!) -- являются комментариями.

```
# Эта строка - комментарий.
```

Комментарии могут располагаться и в конце строки с исполняемым кодом.

```
echo "Далее следует комментарий." # Это комментарий.
```

Комментариям могут предшествовать пробелы (пробел, табуляция).

```
        # Перед комментарием стоит символ табуляции.
```

Само собой разумеется, экранированный символ # в операторе echo не воспринимается как начало комментария. Более того, он может использоваться в операциях подстановки параметров и в константных числовых выражениях.

```
echo "Символ # не означает начало комментария."
echo 'Символ # не означает начало комментария.'
echo Символ \# не означает начало комментария.
echo А здесь символ # означает начало комментария.
echo ${PATH#*:}       # Подстановка -- не комментарий.
echo $(( 2#101011 ))  # База системы счисления -- не
комментарий.
```

Кавычки " ' и \ экранируют действие символа #.

Разделитель команд. «Точка-с-запятой» Позволяет записывать две и более команд в одной строке.

```
echo hello; echo there
```

Следует отметить, что символ ";" иногда так же, как и # необходимо экранировать.

Двойные кавычки. В строке "STRING", ограниченной двойными кавычками, не выполняется интерпретация большинства служебных символов, которые могут находиться в строке.

Одинарные кавычки. 'STRING' экранирует все служебные символы в строке STRING. Это более строгая форма экранирования.

Запятая. Оператор запятая используется для вычисления серии арифметических выражений. Вычисляются все выражения, но возвращается результат последнего выражения.

```
let "t2 = ((a = 9, 15 / 3))"  # Присваивает значение переменной "a" и вычисляет "t2".
```

escape. «обратный слэш» Комбинация \X "экранирует" символ X. Аналогичный эффект имеет комбинация с "одинарными кавычками", т.е. 'X'. Символ \ может использоваться для экранирования кавычек " и '.

Разделитель, используемый в указании пути к каталогам и файлам. «слэш» Отделяет элементы пути к каталогам и файлам (например, /home/bozo/projects/Makefile).

В арифметических операциях он является оператором деления.

Подстановка команд. «Обратные кавычки» могут использоваться для записи в переменную команды `command`.

Переменные — это одна из основ любого языка программирования. Они участвуют в арифметических операциях, в синтаксическом анализе строк и совершенно необходимы для абстрагирования каких-либо величин с помощью символических имен. Физически переменные представляют собой ни что иное как участки памяти, в которые записана некоторая информация.

Подстановка переменных

Когда интерпретатор встречает в тексте сценария имя переменной, то он вместо него подставляет значение этой переменной. Поэтому ссылки на переменные называются подстановкой переменных.

Необходимо всегда помнить о различиях между именем переменной и ее значением. Если variable1 — это имя переменной, то $variable1 — это ссылка на ее значение. "Чистые" имена переменных, без префикса $, могут использоваться только при объявлении переменной, при присваивании переменной некоторого значения, при удалении (сбросе), при экспорте и в особых случаях - когда переменная представляет собой название сигнала. Присваивание может производится с помощью символа = (например, var1=27), инструкцией read и в заголовке цикла (for var2 in 1 2 3).

Пример. Присваивание значений переменным и подстановка значений переменных

```
#!/bin/bash

# Присваивание значений переменным и подстановка значений переменных

a=375
hello=$a

#-------------------------------------------------------------------
# Использование пробельных символов
```

```
  # с обеих сторон символа "=" присваивания недопустимо.

  #  Если записать "VARIABLE =value",
  #+ то интерпретатор попытается выполнить команду
"VARIABLE" с параметром "=value".

  #  Если записать "VARIABLE= value",
  #+ то интерпретатор попытается установить переменную
окружения "VARIABLE" в ""
  #+ и выполнить команду "value".
  #-----------------------------------------------------
---------------------


  echo hello     # Это не ссылка на переменную, выведет
строку "hello".

  echo $hello
  echo ${hello} # Идентично предыдущей строке.

  echo "$hello"
  echo "${hello}"

  echo

  hello="A B  C   D"

  echo $hello   # A B C D

  echo "$hello" # A B  C   D

  # Здесь видны различия в выводе echo $hello и echo
"$hello".
  # Заключение ссылки на переменную в кавычки сохраняет
пробельные символы.

  echo
```

```
  echo '$hello'  # $hello

  # Внутри одинарных кавычек не производится подстановка
значений переменных,
  #+  "$" интерпретируется как простой символ.

  # Существуют различия между этими типами кавычек.


  hello=    # Запись пустого значения в переменную.
  echo "\$hello (пустое значение) = $hello"
  #  Запись пустого значения — это не то же самое,
  #+ что сброс переменной, хотя конечный результат --
тот же.

  # ------------------------------------------------
-----------

  #  Допускается присваивание нескольких переменных в
одной строке,
  #+ если они отделены пробельными символами.
  #  Внимание! Это может снизить читабельность сценария
и оказаться непереносимым.

  var1=variable1  var2=variable2  var3=variable3
  echo
  echo "var1=$var1   var2=$var2  var3=$var3"

  # Могут возникнуть проблемы с устаревшими версиями
"sh".

  # ------------------------------------------------
-----------

  echo; echo

  numbers="один два три"
```

```
  other_numbers="1 2 3"

  # Если в значениях переменных встречаются пробелы,
  # то использование кавычек обязательно.

  echo "numbers = $numbers"
  echo "other_numbers = $other_numbers"   # other_numbers
= 1 2 3
  echo

  echo             "uninitialized_variable             =
$uninitialized_variable"

  # Неинициализированная переменная содержит "пустое"
значение.

  uninitialized_variable=
  #  Объявление неинициализированной переменной
  #+ (то же, что и присваивание пустого значения, см.
выше).

  echo             "uninitialized_variable             =
$uninitialized_variable"

  # Переменная содержит "пустое" значение.

  uninitialized_variable=23
   # Присваивание.

  unset uninitialized_variable
   # Сброс.

  echo             "uninitialized_variable             =
$uninitialized_variable"
  # Переменная содержит "пустое" значение.
  echo
  exit 0
```

Проверка условий

Практически любой язык программирования включает в себя условные операторы, предназначенные для проверки условий, чтобы выбрать тот или иной путь развития событий в зависимости от этих условий. В Bash для проверки условий, имеется команда test и различного вида скобочные операторы, а так же условный оператор if/then.

Конструкции проверки условий

Оператор if/then проверяет -- является ли код завершения списка команд 0 (поскольку 0 означает "успех"), и если это так, то выполняет одну, или более, команд, следующие за словом then.

Существует специальная команда -- [(левая квадратная скобка). Она является синонимом команды test, и является встроенной командой (т.е. более эффективной, в смысле производительности). Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 -- истина, 1 -- ложь).

Начиная с версии 2.02, Bash предоставляет в распоряжение программиста конструкцию [[...]] расширенный вариант команды test, которая выполняет сравнение способом более знакомым программистам, пишущим на других языках программирования. Обратите внимание: [[-- это зарезервированное слово, а не команда.

Bash исполняет [[$a -lt $b]] как один элемент, который имеет код возврата.

Круглые скобки ((...)) и предложение let ... так же возвращают код 0, если результатом арифметического выражения является ненулевое значение. Таким образом, арифметические выражения могут участвовать в операциях сравнения.

Предложение let "1<2" возвращает 0 (так как результат сравнения "1<2" -- "1", или "истина")

((0 && 1)) возвращает 1 (так как результат операции "0 && 1" -- "0", или "ложь")

Циклы и ветвления

Управление ходом исполнения - один из ключевых моментов структурной организации сценариев на языке командной оболочки. Циклы и переходы являются теми инструментальными средствами, которые обеспечивают управление порядком исполнения команд.

Цикл — это блок команд, который исполняется многократно до тех пор, пока не будет выполнено условие выхода из цикла.

Циклы for

for (in) Это одна из основных разновидностей циклов. И она значительно отличается от аналога в языке C.

```
for arg in [list]
do
 команда(ы)...
done
```

Пример. Простой цикл for

```
#!/bin/bash
# Список планет.

for planet in Меркурий Венера Земля Марс Юпитер Сатурн
Уран Нептун Плутон
do
  echo $planet
done

echo

# Если 'список аргументов' заключить в кавычки, то он
будет восприниматься как единственный аргумент.
for planet in "Меркурий Венера Земля Марс Юпитер Сатурн
Уран Нептун Плутон"
do
  echo $planet
done
```

```
exit 0
```

while

Оператор while проверяет условие перед началом каждой итерации и если условие истинно (если код возврата равен 0), то управление передается в тело цикла. В отличие от циклов for, циклы while используются в тех случаях, когда количество итераций заранее не известно.

```
while [condition]
do
 command...
done
```

Как и в случае с циклами for/in, при размещении ключевого слова do в одной строке с объявлением цикла, необходимо вставлять символ ";" перед do.

```
while [condition] ; do
```

Пример. Простой цикл while

```
#!/bin/bash

var0=0
LIMIT=10

while [ "$var0" -lt "$LIMIT" ]
do
  echo -n "$var0 "       # -n подавляет перевод строки.
var0=`expr $var0 + 1`# допускается var0=$(($var0+1)).
done
echo

exit 0
```

until

Оператор цикла until проверяет условие в начале каждой итерации, но в отличие от while итерация возможна только в том случае, если условие ложно.

```
until [condition-is-true]
do
 command...
done
```

Оператор until проверяет условие завершения цикла ПЕРЕД очередной итерацией, а не после, как это принято в некоторых языках программирования.

Как и в случае с циклами for/in, при размещении ключевого слова do в одной строке с объявлением цикла, необходимо вставлять символ ";" перед do.

```
until [condition-is-true] ; do
```

Пример. Цикл until

```
#!/bin/bash

until [ "$var1" = end ] # Проверка условия производится
в начале итерации.
do
  echo "Введите значение переменной #1 "
  echo "(end - выход)"
  read var1
  echo "значение переменной #1 = $var1"
done

exit 0
```

Внутренние команды

Внутренняя команда — это команда, которая встроена непосредственно в Bash. Команды делаются встроенными либо из соображений производительности - встроенные команды исполняются быстрее, чем внешние, которые, как правило, запускаются в дочернем процессе, либо из-за необходимости прямого доступа к внутренним структурам командного интерпретатора.

Ввод/вывод

echo выводит (на stdout) выражение или содержимое переменной.

```
echo Hello
echo $a
printf
```

printf - команда форматированного вывода, расширенный вариант команды echo и ограниченный вариант библиотечной функции printf() в языке C, к тому же синтаксис их несколько отдичается друг от друга.

```
printf format-string... parameter...
```

read "Читает" значение переменной с устройства стандартного ввода - stdin, в интерактивном режиме это означает клавиатуру. Ключ - a позволяет записывать значения в массивы.

Пример. Ввод значений переменных с помощью read

```
#!/bin/bash

echo -n "введите значение переменной 'var1': "
# Ключ -n подавляет вывод символа перевода строки.

read var1
# Обратите внимание -- перед именем переменной отсутствует символ '$'.
```

```
  echo "var1 = $var1"


  echo

  # Одной командой 'read' можно вводить несколько
переменных.

  echo -n "дите значения для переменных 'var2' и 'var3'
(через пробел или табуляцию): "
  read var2 var3
  echo "var2 = $var2      var3 = $var3"

  # Если было введено значение только одной переменной,
то вторая останется "пустой".

  exit 0
```

ЗАПУСК СЦЕНАРИЯ

В простейшем случае, скрипт — это ни что иное, как простой список команд системы, записанный в файл. Создание скриптов поможет сохранить время и силы, которые тратятся на ввод последовательности команд всякий раз, когда необходимо их выполнить.

Если файл сценария начинается с последовательности #!, которая в мире UNIX называется sha-bang, то это указывает системе какой интерпретатор следует использовать для исполнения сценария. Это двухбайтовая последовательность, или - специальный маркер, определяющий тип сценария, в данном случае - сценарий командной оболочки. Более точно, sha-bang определяет интерпретатор, который вызывается для исполнения сценария, это может быть командная оболочка (shell), иной интерпретатор или утилита.

```
#!/bin/sh
#!/bin/bash
#!/usr/bin/perl
#!/usr/bin/tcl
#!/bin/sed -f
#!/usr/awk -f
```

Каждая, из приведенных выше сигнатур, приводит к вызову различных интерпретаторов, будь то /bin/sh - командный интерпретатор по умолчанию (bash для Linux-систем), либо иной. При переносе сценариев с сигнатурой #!/bin/sh на другие UNIX системы, где в качестве командного интерпретатора задан другой shell, можно лишиться некоторых особенностей, присущих bash. Поэтому такие сценарии должны быть POSIX совместимыми.

Обратите внимание на то, что сигнатура должна указывать правильный путь к интерпретатору, в противном случае вы получите сообщение об ошибке - как правило это "Command not found".

Сигнатура #! может быть опущена, если не используются специфичные команды.

Интерпретатор, в свою очередь, воспринимает эту строку как комментарий, поскольку она начинается с символа #.

Если в сценарии имеются еще такие же строки, то они воспринимаются как обычный комментарий.

```
#!/bin/bash

echo "Первая часть сценария."
a=1

#!/bin/bash
# Это *НЕ* означает запуск нового сценария.

echo "Вторая часть сценария."
echo $a  # Значение переменной $a осталось равно 1.
```

Запустить сценарий можно командой sh scriptname или bash scriptname. (Не рекомендуется запуск сценария командой sh <scriptname>, поскольку это запрещает использование устройства стандартного ввода stdin в скрипте). Более удобный вариант - сделать файл скрипта исполняемым, командой chmod.

Это:

chmod 555 scriptname

(выдача прав на чтение/исполнение любому пользователю в системе)

или

chmod +rx scriptname

(выдача прав на чтение/исполнение любому пользователю в системе)

chmod u+rx scriptname

(выдача прав на чтение/исполнение только "владельцу" скрипта)

Если сделаеть файл сценария исполняемым, можно запустить его такой командой ./scriptname. Если, при этом, текст сценария начинается

с корректной сигнатуры ("sha-bang"), то для его исполнения будет вызван соответствующий интерпретатор.

И наконец, завершив отладку сценария, можно поместить его в каталог /usr/local/bin (естественно, что для этого должны быть права root), чтобы сделать его доступным для всех пользователей системы. После этого сценарий можно вызвать, просто напечатав название файла в командной строке и нажав клавишу Enter.

ЗАДАНИЕ НА ДОМАШНЮЮ РАБОТУ

1. Изучить теоретический материал и составить алгоритм работы сценария, согласно варианту задания.
2. Создать текстовый файл и написать в нем код сценария, обеспечивающий требуемый функционал.
3. Запустить и протестировать работоспособность и наличие требуемой функциональности сценария.

Варианты заданий:

Вариант 1

Написать скрипт для реализации простого калькулятора (сложение, вычитание, умножение, деление).

В программе использовать меню. Результат действий сообщить в письме для root.

Вариант 2

Написать скрипт для перевода чисел из одной системы счисления в другую.

Для выбора системы счисления использовать меню. Результат записать в файл в формате: исходное число – исходная система счисления = результирующее число – результирующая система счисления.

Вариант 3

Написать скрипт для разложения числа на простые множители.

Число и результат операции отправить в письме для root. Число пользователь вводит либо с клавиатуры, либо считывает из файла. Для выбора использовать меню

Вариант 4

Написать скрипт для генерации имен N файлов в разных форматах.

Для выбора формата имени использовать меню. Сгенерированные имена записать в файл и сообщить об этом в письме для root.

Вариант 5

Написать скрипт для работы с массивами. Предусмотреть сортировку (по возрастанию и убыванию), поиск и создание массива.

Для выполнения скрипта использовать меню. Каждое действие и его результат сохранять в файл.

Вариант 6

Написать скрипт для изменения имен файлов в текущем каталоге или выбранным пользователем с верхнего регистра на нижний и наоборот.

Создать файл и записать в него количество измененных файлов.

Вариант 7

Написать скрипт для преобразования десятичных чисел в римскую систему счисления.

Результат вывести на экран и сохранить в файл в формате: исходное число = результирующее число.

Вариант 8

Написать скрипт для поиска файла программы по идентификатору процесса.

Вывести список программ и предложить пользователю ввести идентификатор. Входные и выходные данные программы послать в письме к root.

Вариант 9

Написать скрипт для подсчета числа дней между двумя датами.

Диапазон дат задается пользователем либо с клавиатуры, либо находится в файле. Реализовать меню для программы и сообщить входные и выходные данные в письме к root.

Вариант 10

Написать скрипт для удаления из текущего каталога всех файлов размером меньше N байт.

Размер ввести с клавиатуры, и отправить письмо администратору с именами удаленных файлов.

Вариант 11

Переместить из текущего каталога все файлы с именами, начинающимися на A и заканчивающимися на B в каталог, имя которого пользователь задает с клавиатуры.

Отправить письмо администратору с именами перемещенных файлов.

Вариант 12

Написать скрипт, который в текущем каталоге ищет все пустые каталоги и удаляет их.

Отправить письмо администратору с именами удаленных каталогов.

Вариант 13

Написать скрипт, который в расширенном варианте определяет содержимое текущего каталога, заносит его в файл, сжимает любым архиватором и помещает в каталог, который укажет пользователь.

Имя каталога и файла пользователь вводит с клавиатуры.

Вариант 14

Создать базу данных с информацией о пользователях: идентификатор, имя пользователя, имя учетной записи, группа.

Используя меню добавить пользователя в базу данных, редактировать информацию о нем, удалить запись в базе данных и обнулить базу данных, оставив только заголовок таблицы. Информацию сохранять в файл.

Вариант 15

Написать скрипт для поиска файлов языка C++ в текущем каталоге и их компиляции.

Имена файлов с исходным кодом записать в файл с именем в формате: исходный код + дата + время.

Вариант 16

Написать скрипт, в котором с использованием меню создается папка, файл, мягкая и жесткая ссылки, происходит перемещение и копирование файла, архивация и сжатие архива, а также удаление ссылок, файлов и папок и т.д. Предусмотреть отображение содержимого каталога.

Вариант 17

Написать скрипт, который в файле smb.conf добавляет запись о новом ресурсе или удаляет старую.

Для выбора использовать меню. Параметры при добавлении вводит пользователь, если пользователь их не вводит система предлагает свои по умолчанию.

Послать письмо администратору о добавленном или удаленном ресурсе.

Вариант 18

Сгенерировать N файлов со случайными именами, наполнить их случайным содержимым, поместить в архив и сжать.

Написать об этом письмо администратору сообщив названия файлов и их размер.

Вариант 19

Написать скрипт для установки удаленного соединения, просмотреть все открытые доступные ресурсы, занести эту информацию файл.

Администратору отправить письмо сообщим название файла, в котором содержится информация. Имя файла должно быть в формате: имя ресурса + дата + время.

Вариант 20

Написать скрипт, реализующий калькулятор с основными арифметическими функциями и функциями сравнения. Для реализации использовать меню.

Результаты записывать в файл с именем в формате: дата + время.

Вариант 21

Написать скрипт, для получения списка процессов, их завершения и выключения системы (указать причину выключения).

Отослать предварительно письмо администратору с датой и временем и причиной выключения системы.

Вариант 22

Написать скрипт, который определяет в текущем каталоге файл максимального размера удаляет его и все файлы, начинающиеся на тот же символ. Если таких файлов нет, то создать каталог, имя которого пользователь вводит с клавиатуры.

Вариант 23

Написать скрипт, который производит архивацию всех временных файлов в каталоге.

В письме администратору сообщить количество таких файлов, их общий объем и время архивации.

Вариант 24

Установить соединение с сервером и скопировать на него все файлы из текущего каталога размером менее 5 Кб предварительно поместив их в архив.

По завершении закрыть соединение с сервером.

Вариант 25

Написать скрипт, в котором пользователь вводит имя каталога для поиска, производится поиск всех файлов размером от 1 Кб до 10 Кб, они помещаются в архив и сжимаются.

В отдельный файл занести всю информацию о найденных файлах.

Вариант 26

Написать скрипт, в котором с использованием меню пользователю предлагается сменить пользователя, перезагрузить и выключить компьютер.

Предусмотреть возможность подтверждения действия.

В административных целях необходимо занести в системный журнал информацию о дате и времени, пользователе и выполненном действии.

Вариант 27

Написать скрипт, который ищет в текущем каталоге файлы, имена которых вводит пользователь, перемещает их в папку, имя которой вводит пользователь. Если пользователь для имени папки ввел 0, то создается папка с именем дата + время.

Вариант 28

Написать скрипт для нахождения определенного интеграла по формуле Симпсона.

Диапазон задает пользователь с клавиатуры, функции предопределены в программе и выбираются при помощи меню. Результат заносится в файл в виде: функция, диапазон и результат. На экран выводится только результат.

Вариант 29

Написать скрипт, который формирует файлы с информацией о текущем каталоге в упорядоченном виде: по алфавиту, по размеру и по дате создания.

Формат имени файла имеет размер тип + дата + время.

По запросу пользователя необходимо отобразить изменения в каталоге, основанные на созданных файлах: показать созданные, удаленные файлы и файлы, размер которых был изменен и на сколько.

Вариант 30

Написать скрипт для решения квадратного уравнения.

Пользователь либо вводит значения с клавиатуры, либо значения считываются из файла. Результат заносится в файл, выводится на экран и отправляется письмо root с коэффициентами уравнения и найденным решением. В программе использовать меню.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Опишите назначение Shell-скриптов.
2. Опишите термин BASH.
3. Опишите назначение символа «#», и приведите примеры его использования.
4. Опишите для чего необходимо экранирование символов.
5. Назовите различия одинарных и двойных кавычек.
6. Опишите понятие переменная.
7. Приведите пример кода с условными операторами.
8. Опишите принцип работы с переменными в Shell.
9. Приведите пример кода с оператором цикла while.
10. Опишите понятие внутренняя команда.
11. Назовите команды для вывода информации.
12. Опишите понятие sha-bang.
13. Назовите способы запуска сценария.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение домашней работы отводится 10 академических часов: 9 часов на выполнение и сдачу домашней работы и 1 час на подготовку отчета.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)):

- титульный лист;
- цель, задачи, формулировка задания;
- блок-схема работы сценария;
- листинг разработанного Shell-сценария;
- описание используемых операторов, команд и их параметров;
- результаты работы Shell-сценария;
- выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Вирт, Н. Разработка операционной системы и компилятора. Проект Оберон [Электронный ресурс] / Н. Вирт, Ю. Гуткнехт. — Москва: ДМК Пресс, 2012. 560 с. Режим доступа: https://e.lanbook.com/book/39992.

2. Войтов, Н.М. Основы работы с Linux. Учебный курс [Электронный ресурс]: учебное пособие / Н.М. Войтов. — Москва : ДМК Пресс, 2010. — 216 с. — Режим доступа: URL: https://e.lanbook.com/book/1198

3. Стащук, П.В. Краткое введение в операционные системы [Электронный ресурс] : учебное пособие / П.В. Стащук. — 3-е изд., стер. — Москва : ФЛИНТА, 2019. — 124 с.— URL: https://e.lanbook.com/book/125385

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Войтов, Н.М. Администрирование ОС Red Hat Enterprise Linux. Учебный курс [Электронный ресурс] : учеб. пособие — Москва: ДМК Пресс, 2011. 192 с. Режим доступа: https://e.lanbook.com/book/1081.

5. Стащук П.В. Администрирование и безопасность рабочих станций под управлением Mandriva Linux: лабораторный практикум. [Электронный ресурс]: учебно-методическое пособие / П.В. Стащук. — 2-е изд., стер. - М: Флинта, 2015. https://e.lanbook.com/book/70397

Электронные ресурсы:

1. Научная электронная библиотека http://eLIBRARY.RU.
2. Электронно-библиотечная система http://e.lanbook.com.
3. Электронно-библиотечная система «Университетская библиотека онлайн» http://biblioclub.ru.
4. Электронно-библиотечная система IPRBook http://www.iprbookshop.ru/
5. Losst - Linux Open Source Software Technologies https://losst.ru