Отчет по лабораторной работе 11

Программирование в командном процессоре ОС UNIX. Командные файлы

Шалыгин Георгий Эдуардович, НФИбд-02-20

Содержание

# Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Объект исследования: система UNIX.

Предмет исследования: программирование в UNIX.

# Техническое обеспечение:

* Характеристики техники: AMD Ryzen 5 3500U 2.1 GHz, 8 GB оперативной памяти, 50 GB свободного места на жёстком диске;
* ОС Windows 10 Home
* Git 2.31.1
* Google Chrome 91.0.4472.19
* VirtualBox 2.0
* CentOS 7

# Условные обозначения и термины:

**Текстовым редактором**(text editor) называют программу, которая предназначена для редактирования (составления и изменения) файлов, содержащих только текст. [1]

**Командный язык** - это язык, на котором пользователь взаимодействует с системой в интерактивном режиме.

**Командный интерпретатор**, интерпретатор командной строки - компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.[3]

Подробнее в [2] и [3].

# Теоретическое введение:

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

– оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций;

– С-оболочка (или csh) — надстройка на оболочкой Борна, использующая Сподобный синтаксис команд с возможностью сохранения истории выполнения команд;

– оболочка Корна (или ksh) — напоминает оболочку С, но операторы управления программой совместимы с операторами оболочки Борна;

– BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек С и Корна (разработка компании Free Software Foundation).

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linuxподобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже. [3]

**Переменные в языке программирования bash**

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Например, команда mark=/usr/andy/bin присваивает значение строки символов /usr/andy/bin переменной mark типа строка символов.

Использование значения, присвоенного некоторой переменной, называется подстановкой. Для того чтобы имя переменной не сливалось с символами, которые могут следовать за ним в командной строке, при подстановке в общем случае используется следующая форма записи: ${имя переменной}. [3]

**Использование арифметических вычислений. Операторы let и read**

Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный.

Команда let берет два операнда и присваивает их переменной. Положительным моментом команды let можно считать то, что для идентификации переменной ей не нужен знак доллара; вы можете писать команды типа let sum=x+7, и let будет искать переменную x и добавлять к ней 7.

Команда read позволяет читать значения переменных со стандартного ввода:

echo “Please enter Month and Day of Birth ?”

read mon day trash [3]

**Командные файлы и функции**

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: bash командный\_файл [аргументы]

Чтобы не вводить каждый раз последовательности символов bash, необходимо изменить код защиты этого командного файла, обеспечив доступ к этому файлу по выполнению. Это может быть сделано с помощью команды chmod +x имя\_файла

**Передача параметров в командные файлы и специальные переменные**

Пусть к командному файлу where имеется доступ по выполнению и этот командный файл содержит следующий конвейер: who | grep $1.

В ходе интерпретации файла командным процессором вместо комбинации символов $1 осуществляется подстановка значения первого и единственного параметра andy.

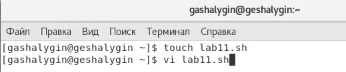
Команда shift позволяет удалять первый параметр и сдвигает все остальные на места предыдущих. При использовании в командном файле комбинации символов $# вместо неё будет осуществлена подстановка числа параметров, указанных в командной строке при вызове данного командного файла на выполнение.

**Оператор цикла for**

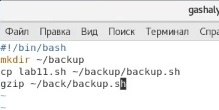
В обобщённой форме оператор цикла for выглядит следующим образом: for имя [in список-значений] do список-команд done. [3]

# Выполнение лабораторной работы

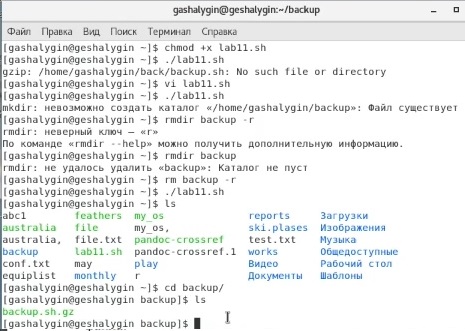
1. Создадим новый файл lab11.sh. (рис. 1)

* 
* Figure 1: Создание нового файла

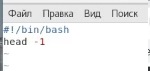
1. Напишем скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге (рис. 2).

* 
* Figure 2: Текст 1 скрипта

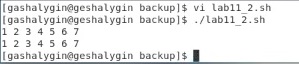
1. Результат работы скрипта, видим архив в нужной директории. (рис. 3).

* 
* Figure 3: Результат создание backup-архива

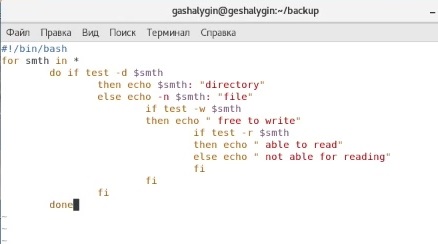
1. Напишем командный файл, который будет последовательно распечатывать значения всех переданных аргументов. (рис. 4)

* 
* Figure 4: Текст 2 скрипта

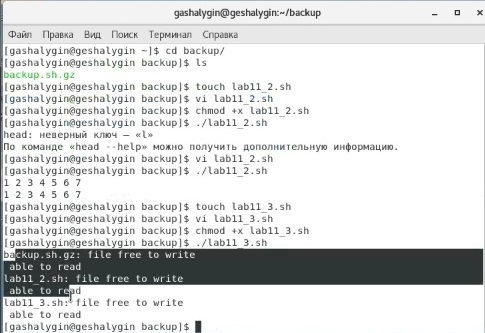
1. Результат работы скрипта. (рис. 5)

* 
* Figure 5: Вывод аргументов в консоль, результат

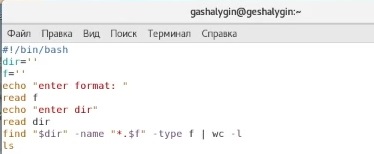
1. Напишем командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога. (рис. 6)

* 
* Figure 6: Текст 3 скрипта

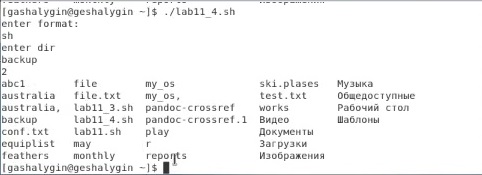
1. Результат работы скрипта. (рис. 7)

* 
* Figure 7: Описание содержимого директории, результат

1. Напишем командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки. (рис. 8)

* 
* Figure 8: Текст 4 скрипта

1. Результат работы скрипта, видим два файла - как и должно быть. (рис. 9)

* 
* Figure 9: Количество файлов опр. расширения

# Выводы

В процессе работы над лабораторной работы были изучены основы программирования в оболочке ОС UNIX/Linux, получены навыки написания небольших командных файлов.

# Библиография

1. https://docs.altlinux.org/ru-RU/archive/2.3/html-single/junior/alt-docs-extras-linuxnovice/ch02s10.html
2. http://bourabai.kz/os/shells.htm
3. Д.С. Кулябов, А.В. Королькова / Администрирование локальных систем. Лабораторные работы. — М.: Российский университет дружбы народов, 2017. — 119 с.