Отчет по лабораторной работе №11

Дисциплина: операционные системы

Астраханцева А. А.

Содержание

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# 2 Задание

1. Используя команды getopts grep, написать командный файл, который анализирует командную строку с ключами: – -iinputfile — прочитать данные из указанного файла; – -ooutputfile — вывести данные в указанный файл; – -pшаблон — указать шаблон для поиска; – -C — различать большие и малые буквы; – -n — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом -p.
2. Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. Затем программа завершается с помощью функции exit(n), передавая информацию в о коде завершения в оболочку. Команд- ный файл должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено.
3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до 𝑁 (например 1.tmp, 2.tmp, 3.tmp,4.tmp и т.д.). Число файлов, которые необходимо создать, передаётся в аргументы командной строки. Этот же ко- мандный файл должен уметь удалять все созданные им файлы (если они существуют).
4. Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).

# 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 С-подобный 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/Linux-подобных 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.

# 4 Выполнение лабораторной работы

Для начала создадим командный файл №1(рис. [1](#fig:001)).

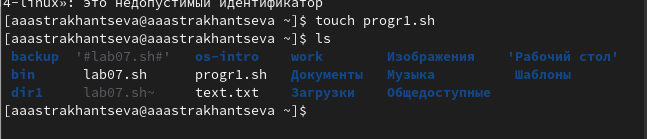


Figure 1: Создание командного файла №1

В созданный файл записывам текст нашей програмы. Используем опреатор case для выбора опций (рис. [2](#fig:002)).

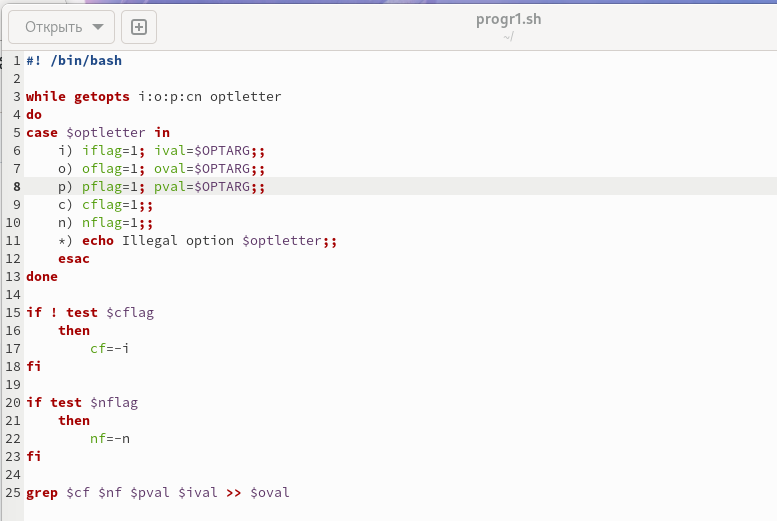


Figure 2: Текст программы №1

Проверяем, что работает корректно (рис. [3](#fig:003)).

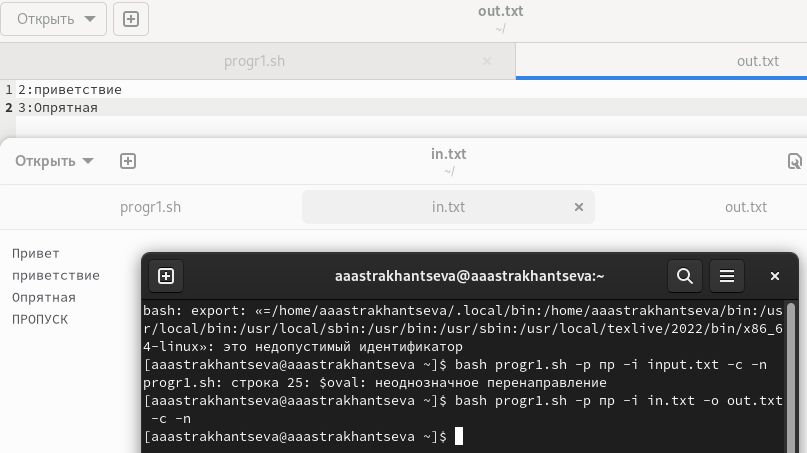


Figure 3: Проверка работа програмы №1

Создаем командный файл №2 и файл для программы на Си (рис. [4](#fig:004)).

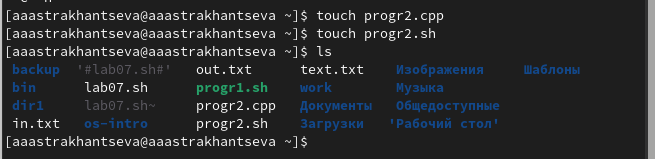


Figure 4: Создание командного файла №2

В созданный файл записывам текст нашей програмы на языке Си(рис. [5](#fig:005)).



Figure 5: Текст программы №2 (Си)

В командный файл записываем текст программы. Используем опреатор case (рис. [6](#fig:006)).

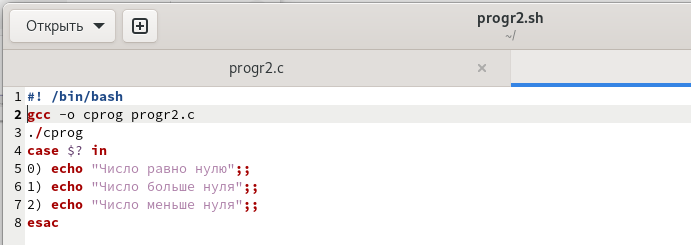


Figure 6: Текст программы №2

Проверяем, что работает корректно (рис. [7](#fig:007)).

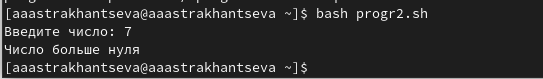


Figure 7: Проверка работа програмы №2

Создаем командный файл №3 (рис. [8](#fig:008)).

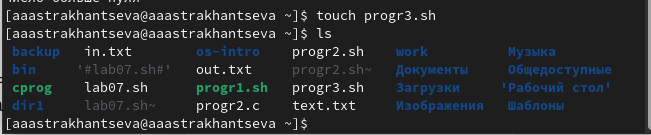


Figure 8: Создание командного файла №3

В командный файл записываем текст программы. Используем цикл for(рис. [9](#fig:009)).



Figure 9: Текст программы №3

Проверяем, что работает корректно (рис. [10](#fig:010)).

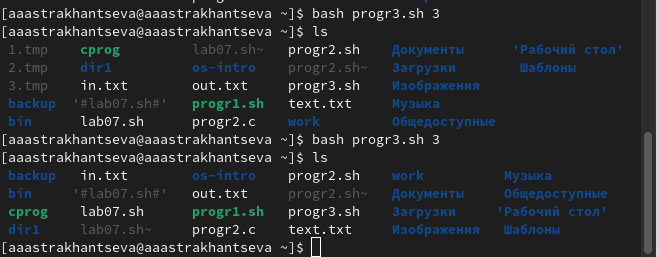


Figure 10: Проверка работа програмы №3

Создаем командный файл №4 (рис. [11](#fig:011)).

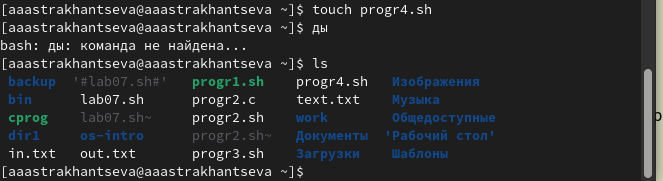


Figure 11: Создание командного файла №4

В командный файл записываем текст программы. Используем команды find и tar(рис. [12](#fig:012)).

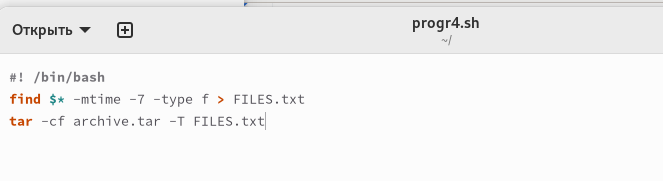


Figure 12: Текст программы №4

Проверяем, что работает корректно (рис. [13](#fig:013)).

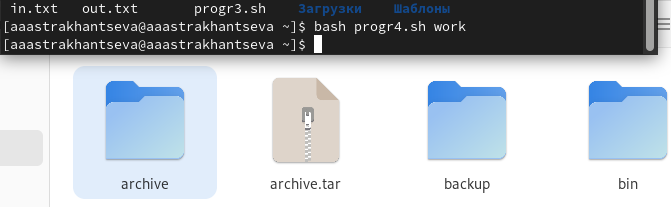


Figure 13: Проверка работа програмы №4

**Контрольные вопросы**

1. Каково предназначение команды getopts?

Команда getopts используется shell-процедурами для разбора позиционных параметров и проверки опций на допустимость. Команда getopts поддерживает все применимые правила стандартного синтаксиса команд

Цепочка\_опций должна содержать флаги, которые будут распознаваться shell-процедурой, использующей getopts. Если за флагом следует двоеточие, то предполагается, что опция имеет аргумент или группу аргументов, которые должны отделяться от флага пробелами или табуляциями.

После каждого вызова getopts помещает следующую опцию в переменную имя, а номер аргумента, который должен быть обработан следующим, - в переменную OPTIND. Всегда, когда запускается shell или shell-процедура, OPTIND получает значение 1.

Если опции требуется аргумент, то getopts помещает его в переменную OPTARG.

Если встретилась некорректная опция, то в переменную имя помещается ?.

Если опции кончились, getopts возвращает ненулевой код завершения. Можно использовать специальную опцию –, чтобы отметить конец опций.

По умолчанию команда getopts разбирает позиционные параметры вызвавшей ее shell-процедуры, но если указать дополнительные аргументы, то getopts будет разбирать их.

1. Какое отношение метасимволы имеют к генерации имён файлов?

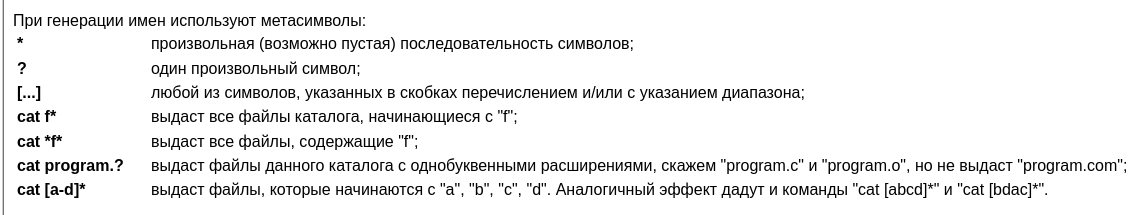


Figure 14: Метасимволы

1. Какие операторы управления действиями вы знаете?

Для решения подобных задач язык программирования bash предоставляет возможность использовать такие управляющие конструкции, как for, case, if и while. С точки зрения командного процессора эти управляющие конструкции являются обычными командами и могут использоваться как при создании командных файлов, так и при работе в интерактивном режиме. Команды, реализующие подобные конструкции, по сути, являются операторами языка программирования bash. Поэтому при описании языка программирования bash термин оператор будет использоваться наравне с термином команда. Команды ОС UNIX возвращают код завершения, значение которого может быть использовано для принятия решения о дальнейших действиях. Команда test, например, создана специально для использования в командных файлах. Единственная функция этой команды заключается в выработке кода завершения.

1. Какие операторы используются для прерывания цикла?

Команда break завершает выполнение цикла, а команда continue завершает данную итерацию блока операторов. Команда break полезна для завершения цикла while в ситуациях, когда условие перестаёт быть правильным. Команда continue используется в ситуациях, когда больше нет необходимости выполнять блок операторов, но вы можете захотеть продолжить проверять данный блок на других условных выражениях.

1. Для чего нужны команды false и true?

Следующие две команды ОС UNIX используются только совместно с управляющими конструкциями языка программирования bash: это команда true, которая всегда возвращает код завершения, равный нулю (т.е. истина), и команда false, которая всегда возвращает код завершения, не равный нулю (т. е. ложь).

Что означает строка if test -f man$s/$i.$s, встреченная в командном файле?

Строка if test -f mani.s/s и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).

Объясните различия между конструкциями while и until.

Выполнение оператора цикла while сводится к тому, что сначала выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), выполняется последовательность команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово do, после чего осуществляется безусловный переход на начало оператора цикла while. Выход из цикла будет осуществлён тогда, когда последняя выполненная команда из последовательности команд (операторов), которую задаёт список-команд в строке, содержащей служебное слово while, возвратит ненулевой код завершения (ложь). При замене в операторе цикла while служебного слова while на until условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное. В остальном оператор цикла while и оператор цикла until идентичны.

# 5 Выводы

Я изучила основы программирования в оболочке ОС UNIX, научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.