Отчет

Лабораторная работа №11

Подоляк Иван НПМбд-02-21

Содержание

# Цель работы

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

# 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.

1. 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. Командный файл должен вызывать эту программу и, проанализировав с помощью команды $?, выдать сообщение о том, какое число было введено.
2. 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 командный файл должен уметь удалять все созданные им файлы (если они существуют).
3. 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).

# 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.

# Выполнение лабораторной работы

1. Создаю первый исполняемый файл *1.sh* и открываю редактор *emacs*. (рис. [-@fig:001])

создание файла

создание файла

1. Используя команды getopts grep, пишу 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. (рис. [-@fig:002]) (рис. [-@fig:003])

написание скрипта

написание скрипта

написание скрипта

написание скрипта

1. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:004])

проверка первого файла

проверка первого файла

1. Создаю второй исполняемый файл *2.sh* и *2.c* и открываю редактор *emacs*. (рис. [-@fig:005])

создание файла

создание файла

1. Пишу 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. (рис. [-@fig:006]) Командный файл вызывает эту программу и, проанализировав с помощью команды $?, выдает сообщение о том, какое число было введено. (рис. [-@fig:007])

написание 2.c

написание 2.c

написание 2.sh

написание 2.sh

1. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:008])

проверка второго файла

проверка второго файла

1. Создаю третий исполняемый файл *3.sh* и открываю редактор *emacs*. (рис. [-@fig:009])

создание файла

создание файла

1. Пишу 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 командный файл должен уметь удалять все созданные им файлы (если они существуют). (рис. [-@fig:010])

написание скрипта

написание скрипта

1. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:011])

проверка третьего файла

проверка третьего файла

1. Создаю четвертый исполняемый файл *4.sh* и открываю редактор *emacs*. (рис. [-@fig:012])

создание файла

создание файла

1. Пишу командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицирую его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использую команду find). (рис. [-@fig:013])

написание скрипта

написание скрипта

1. Добавляю право на выполнение файла и проверяю его работу. (рис. [-@fig:014])

проверка четвертого файла

проверка четвертого файла

# Выводы

Выполняя данную лабораторную работу я изучил основы программирования в оболочке ОС UNIX/Linux и научилcя писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

# Список литературы