Отчёт по лабораторной работе №13

Операционные системы

Луангсуваннавонг Сайпхачан

Содержание

Список иллюстраций

Список таблиц

# 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. Командный 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 командный 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

Сначала я создаю файл program5.sh и открываю его в текстовом редакторе для редактирования. Затем я добавляю код для программы, которая будет искать в указанном файле нужные строки, используя команды getopts и grep. После завершения редактирования я сохраняю и закрываю файл. (рис. 1 и рис. 2)

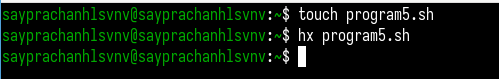


Рис. 1: Создание файла

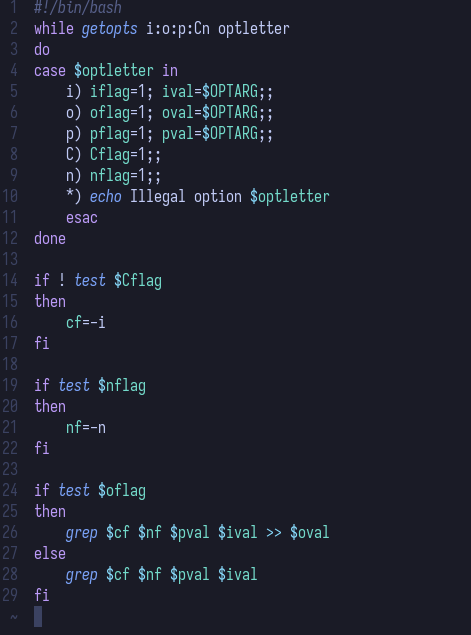


Рис. 2: Код программы

Код программы:

#!/bin/bash  
while getopts i:o:p:Cn optletter  
do  
case $optletter in  
 i) iflag=1; ival=$OPTARG;;  
 o) oflag=1; oval=$OPTARG;;  
 p) pflag=1; pval=$OPTARG;;  
 C) Cflag=1;;  
 n) nflag=1;;  
 \*) echo Illegal option $optletter  
 esac  
done  
  
if ! test $Cflag  
then  
 cf=-i  
fi  
  
if test $nflag  
then  
 nf=-n  
fi  
  
if test $oflag  
then  
 grep $cf $nf $pval $ival >> $oval  
else  
 grep $cf $nf $pval $ival  
fi

Я создаю текстовый файл log.txt для сохранения результата работы программы при выполнении команды, которая требует записи результата в текстовый файл. (рис. 3)

Рис. 3: Создание файла

Рис. 3: Создание файла

Я также создаю тестовый файл с именем test.txt. Я записываю в файл некоторый текст, так как этот файл будет использоваться для тестирования поиска в программе. (рис. 4)

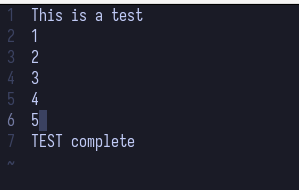


Рис. 4: Создание файла

Затем я даю файлу разрешение на выполнение. Я запускаю программу: сначала, с выводом результата на экран. (рис. 5)

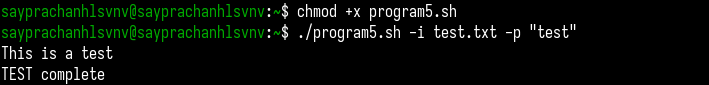


Рис. 5: Запуск программы

Во второй раз я запускаю программу с выводом результата в созданный текстовый файл, добавив опцию -o с указанием имени файла (log.txt). Я открываю текстовый файл log.txt, и там есть строки, что означает, что программа работает корректно. (рис. 6 и рис. 7)

Рис. 6: Запуск программы

Рис. 6: Запуск программы

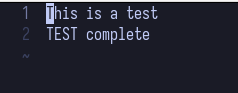


Рис. 7: Создание файла

Затем я создаю файл program6.sh, а также C программу, так как во второй задаче нужно использовать оба файла: bash-скрипт и C программу. C Программа будет запрашивать число и определять, больше ли оно нуля, меньше или равно нулю, после чего завершаться с помощью функции exit(n). Bash-файл будет вызывать программу и выводить сообщение. (рис. 8)

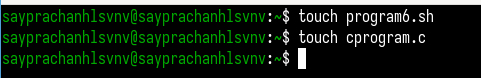


Рис. 8: Создание файлов

В C-программе я ввожу код, который запрашивает ввод числа, затем определяет его значение относительно нуля, и завершает выполнение с помощью функции exit. (рис. 9)

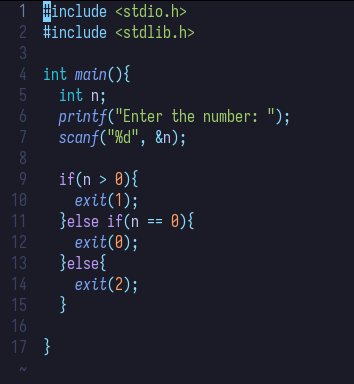


Рис. 9: Код программы

Код программы:

#include <stdio.h>  
#include <stdlib.h>  
  
int main(){  
 int n;  
 printf("Enter the number: ");  
 scanf("%d", &n);  
  
 if(n > 0){  
 exit(1);  
 }else if(n == 0){  
 exit(0);  
 }else{  
 exit(2);  
 }  
   
}

В bash-файле program6.sh я добавляю команду для компиляции программы на C, затем её вызов, а после этого, конструкцию case для вывода соответствующего сообщения на экран. (рис. 10)

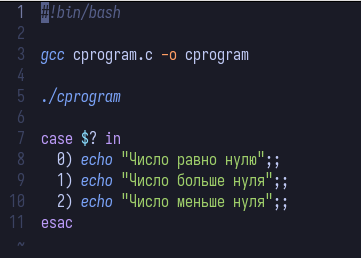


Рис. 10: Код программы

Код программы:

#!bin/bash  
  
gcc cprogram.c -o cprogram  
  
./cprogram  
  
case $? in  
 0) echo "Число равно нулю";;  
 1) echo "Число больше нуля";;  
 2) echo "Число меньше нуля";;  
esac

Я даю файлу разрешение на выполнение и запускаю программу. Ввожу число и программа корректно выводит текст. (рис. 11)

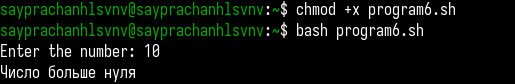


Рис. 11: Запуск программы

Я создаю bash-файл program7.sh и открываю его в текстовом редакторе. В этой программе создаётся указанное количество файлов. Количество создаваемых файлов передаётся в аргументах командной строки. Если файлы уже существуют, программа их удаляет. (рис. 12)

Рис. 12: Создание файла

Рис. 12: Создание файла

Я добавляю код в файл program7.sh. Программа работает в цикле for, проверяя, существует ли файл с заданным именем. Если файл существует, он удаляется, иначе создаётся новый. Количество создаваемых файлов определяется переданным числом. (рис. 13)

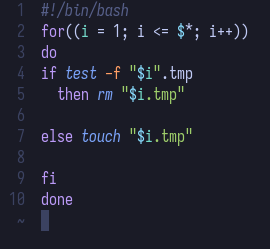


Рис. 13: Код программы

Код программы:

#!/bin/bash  
for((i = 1; i <= $\*; i++))  
do  
if test -f "$i".tmp  
 then rm "$i.tmp"  
  
else touch "$i.tmp"  
  
fi  
done

Затем я запускаю программу, ввожу число 3 , программа создаёт 3 файла. Затем я снова запускаю команду и программа удаляет созданные ранее 3 файла, так как они уже существуют, что подтверждает корректность её работы. (рис. 14)

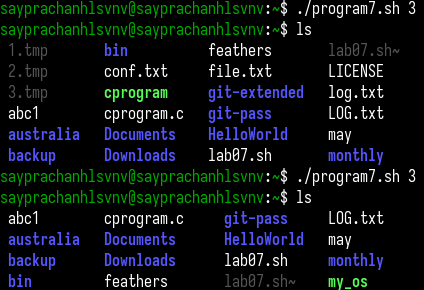


Рис. 14: Запуск программы

Затем я создаю файл program8.sh и открываю его для редактирования. В этой программе создаётся архив в указанной директории, и в архив включаются только те файлы, которые были изменены менее недели назад. (рис. 15)

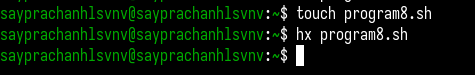


Рис. 15: Создание файла

Я создаю тестовую директорию и несколько текстовых файлов для тестирования этой программы. (рис. 16)

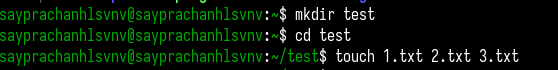


Рис. 16: Создание каталога и файлов

В файле program8.sh я добавляю код, который находит файлы, изменённые менее недели назад, с помощью команды find с опцией -mtime, за которой следует количество дней. Результат сохраняется в LOG.txt, который создаётся после запуска программы. Затем с помощью команды tar создаётся архив директории и файлов, перечисленных в LOG.txt. (рис. 17)

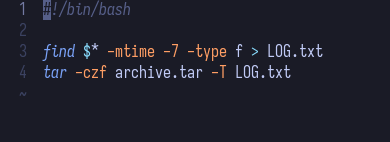


Рис. 17: Код программы

Код программы:

#!/bin/bash  
  
find $\* -mtime -7 -type f > LOG.txt  
tar -czf archive.tar -T LOG.txt

После завершения редактирования я даю файлу разрешение на выполнение и запускаю программу. (рис. 18) Ввожу путь к нужной директории. В результате программа создаёт архив директории и файлов. (рис. 19) Это подтверждает, что программа работает корректно, а в файле LOG.txt отображается список файлов, включённых в архив. (рис. 20)

Рис. 18: Запуск программы

Рис. 18: Запуск программы



Рис. 19: Созданный архив

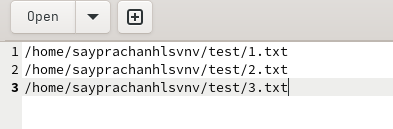


Рис. 20: Список файлов

# 5 Выводы

Во время этой лабораторной работы я изучил основы программирования в оболочке UNIX. Научился писать более сложные пакетные файлы, используя логические структуры управления и циклы.

# 6 Ответы на контрольные вопросы

1. Каково предназначение команды getopts?

Команда getopts используется в shell-скриптах для разбора переданных аргументов командной строки. Она позволяет обрабатывать опции (например, -f, -v) и их аргументы удобным способом. Это встроенная команда Bash, которая часто используется внутри цикла для последовательного анализа всех переданных опций.

1. Какое отношение метасимволы имеют к генерации имён файлов?

Метасимволы (например, *, ?, [ ]) используются в shell для шаблонного поиска и генерации имён файлов. Это называется глоббинг (file name globbing). Например,* .txt соответствует всем файлам с расширением .txt, а file?.sh — всем файлам вроде file1.sh, fileA.sh.

1. Какие операторы управления действиями вы знаете?

В оболочке Bash есть разные операторы:

&& (логическое И) — выполнить следующую команду, если предыдущая завершилась успешно (код 0).

|| (логическое ИЛИ) — выполнить следующую команду, если предыдущая завершилась с ошибкой.

; — разделяет команды, выполняет их последовательно, независимо от результата.

& — запускает команду в фоновом режиме.

Также можно добавить управляющие конструкции:

if, then, else, elif, fi

case, esac

for, while, until, do, done

select

1. Какие операторы используются для прерывания цикла?

break — немедленно завершает выполнение текущего цикла.

continue — пропускает текущую итерацию и переходит к следующей.

1. Для чего нужны команды false и true?

true — всегда возвращает статус завершения 0 (успех).

false — всегда возвращает статус завершения 1 (ошибка). Они часто используются в условиях и циклах для управления потоком выполнения, например в бесконечном цикле while true.

1. Что означает строка if test -f mani.$s, встреченная в командном файле?

Эта строка проверяет, существует ли обычный файл (-f) с именем, составленным из переменных $s и $i, например: man1/help.1. Условие test -f возвращает true, если файл существует и является обычным (не директорией, не устройством и т.п.).

1. Объясните различия между конструкциями while и until.

while выполняет цикл пока условие истинно (возвращает 0).

until выполняет цикл пока условие ложно (возвращает не 0), то есть до тех пор, пока оно не станет истинным.

# Список литературы

[Лабораторная работа №13](https://esystem.rudn.ru/pluginfile.php/2586878/mod_resource/content/5/011-lab_shell_prog_2.pdf)