

Отчёт по лабораторной работе №13

Операционные системы

Румянцев А.О

4 мая 2024

Российский университет дружбы народов, Москва, Россия

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:
 - `-i`inputfile — прочитать данные из указанного файла;
 - `-o`outputfile — вывести данные в указанный файл;
 - `-р`шаблон — указать шаблон для поиска;
 - `-C` — различать большие и малые буквы;
 - `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-р`.
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.

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;

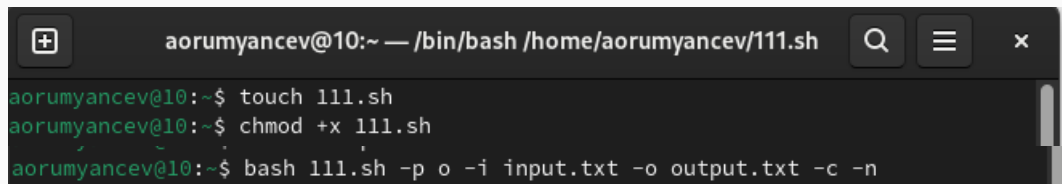
C-оболочка (или csh) — надстройка на оболочке Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;

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

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

Выполнение лабораторной работы

Создаю файл с разрешением на исполнение (рис. 1).



The image shows a terminal window with a dark background. The title bar at the top reads "aorumyancev@10:~ — /bin/bash /home/aorumyancev/111.sh". On the left of the title bar is a square button with a plus sign, and on the right are three buttons: a magnifying glass (search), a hamburger menu (three horizontal lines), and a close button (an 'x'). The terminal content shows three lines of commands and their execution:

```
aorumyancev@10:~$ touch 111.sh
aorumyancev@10:~$ chmod +x 111.sh
aorumyancev@10:~$ bash 111.sh -p o -i input.txt -o output.txt -c -n
```

Командный файл,с командами getopt и grep, который анализирует командную строку с ключами:

-iinputfile -прочитать данные из указанного файла; -ooutputfile - вывести данные в указанный файл -ршаблон - указать шаблон для поиска; -C - различать большие и малые буквы; -n -выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом -р.

```
#!/bin/bash

while getopt 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
```

Результат работы программы в файле output.txt

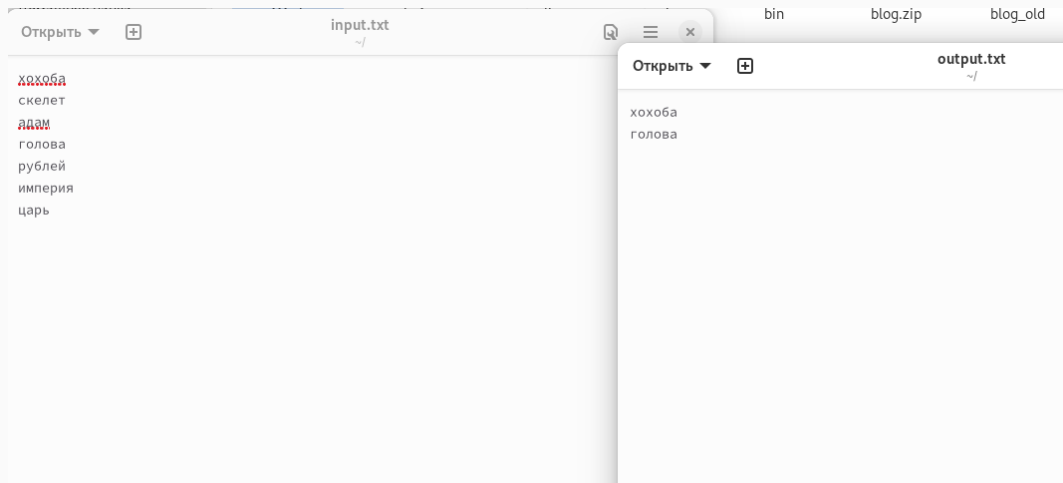


Рис. 2: Результат работы программы

Создаю исполняемый файл для второй программы, также создаю файл 12.c для программы на Си

```
aorумыancev@10:~$ touch 112.sh  
aorумыancev@10:~$ chmod +x 112.sh  
aorумыancev@10:~$ touch 12.cpp  
aorумыancev@10:~$ bash 112.sh
```

Рис. 3: Создание файла

Пишу программу на языке Си, 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

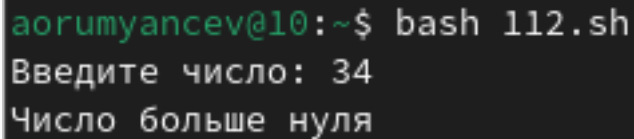
```
#include <stdlib.h>
#include <stdio.h>

int main () {
    int n;
    printf ("Введите число: ");
    scanf ("%d", &n);
    if(n>0){
        exit (1);
    }
    else if (n==0) {
        exit(0);
    }
    else {
        exit(2);
    }
}
```

Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено

```
#!/bin/bash

gcc -o cprog 12.c
./cprog
case $? in
0) echo "Число равно нулю";;
1) echo "Число больше нуля";;
2) echo "Число меньше нуля";;
esac
```

A terminal window with a dark background. The prompt is 'aorunyancev@10:~\$' in green. The command 'bash 112.sh' is entered in white. The program outputs 'Введите число: 34' and 'Число больше нуля' in white.

```
aorunyancev@10:~$ bash 112.sh
Введите число: 34
Число больше нуля
```

Рис. 6: Результат работы программы

```
aorumyancev@10:~$ touch 113.sh  
aorumyancev@10:~$ chmod +x 113.sh  
aorumyancev@10:~$
```

Рис. 7: Создание файла

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)

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

Проверяю, что программа создала файлы и удалила их при соответствующих запросах

```
aorumyancev@10:~$ bash 113.sh 4
aorumyancev@10:~$ ls
111.sh  abc1      feathers  monthly   text.txt  Общедоступные
112.sh  australia fff       mothly.00 Voi        'Рабочий стол'
113.sh  avatar.jpg file1.txt  mouthly.00 work       Шаблоны
12.c    bin       file.txt  my_os     Видео
1.tmp   blog_old  fun       output.txt Документы
2.tmp   blog.zip  input.txt play       Загрузки
3.tmp   conf.txt  main.cpp  reports   Изображения
4.tmp   cprog    may       ski.plases Музыка

aorumyancev@10:~$ bash 113.sh 4
aorumyancev@10:~$ ls
111.sh  bin       file1.txt  mothly.00  text.txt  Музыка
112.sh  blog_old  file.txt  mouthly.00 Voi        Общедоступные
113.sh  blog.zip  fun       my_os     work       'Рабочий стол'
12.c    conf.txt  input.txt output.txt Видео       Шаблоны
abc1    cprog    main.cpp  play       Документы
australia feathers may       reports   Загрузки
avatar.jpg fff      monthly  ski.plases Изображения
```

Создаю исполняемый файл для четвертой программы. Это 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 (использовать команду find)

```
#!/bin/bash
```

```
/home/aorumyancev/114.sh
```

```
find $* -mtime -7 -mtime +0 -type f > FILES.txt
```

```
tar -cf archive.tar -T FILES.txt
```

Рис. 10: Код программы

```
#!/bin/bash
```

```
find $* -mtime -7 -mtime +0 -type f > FILES.txt
```

```
tar -cf archive.tar -T FILES.txt
```

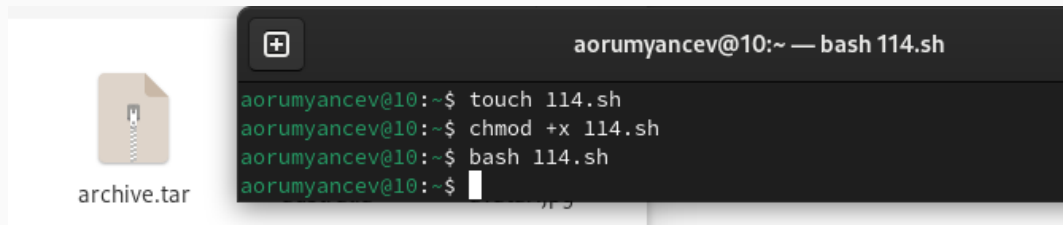


Рис. 11: Результат работы программы

Выводы

При выполнении данной лабораторной работы я изучил основы программирования в оболочке ОС UNIX, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Ответы на контрольные вопросы

1. Каково предназначение команды `getopts`?

Осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable`. Флаги – это опции командной строки, обычно помеченные знаком минус; Например, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -ifile_in.txt -ofile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае: `while getopts o:i:Ltr optletter do case optletter in) o flag = 1; oval =OPTARG;; i) iflag=1; ival=$OPTARG;; L) Lflag=1;; t) tflag=1;; r)`

2. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имён файлов текущего каталога можно использовать следующие символы: `*` – соответствует произвольной, в том числе и пустой строке; `?` – соответствует любому одинарному символу; `[c1-c2]` – соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. Например, `echo *` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .c` – выведет все файлы с последними двумя символами, совпадающими с `.c`. `echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.`. `[a-z]` – соответствует произвольному имени файла в текущем каталоге, начинающемуся с любой строчной буквы латинского алфавита.

3. Какие операторы управления действиями вы знаете?

Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости отрезультатов проверки некоторого условия. 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.

4. Какие операторы используются для прерывания цикла?

Два несложных способа позволяют вам прерывать циклы в оболочке `bash`. 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.

5. 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).

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

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

7. 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.