

Отчет по лабораторной работе №13

Операционные системы

София Андреевна Кудякова

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 4 |
| 2 | Задание | 5 |
| 3 | Теоретическое введение | 7 |
| 4 | Выполнение лабораторной работы | 9 |
| 5 | Выводы | 17 |
| 6 | Ответы на контрольные вопросы | 18 |
| | Список литературы | 22 |

Список иллюстраций

| | | |
|------|--------------------------------------|----|
| 4.1 | Создание файла | 9 |
| 4.2 | Код программы | 10 |
| 4.3 | Результат работы программы | 11 |
| 4.4 | Создание файла | 11 |
| 4.5 | Код программы на Си | 12 |
| 4.6 | Код программы | 13 |
| 4.7 | Результат работы программы | 14 |
| 4.8 | Создание файла | 14 |
| 4.9 | Код программы | 14 |
| 4.10 | Результат работы программы | 15 |
| 4.11 | Код программы | 15 |
| 4.12 | Результат работы программы | 16 |

1 Цель работы

Цель данной лабораторной работы - изучить основы программирования в оболочке ОС UNIX, научиться 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;
 - `-р`шаблон — указать шаблон для поиска;
 - `-С` — различать большие и малые буквы;
 - `-п` — 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. 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 ∞ (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;

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).

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

Создаю файл с разрешением на исполнение (рис. fig. 4.1).

```
sakudyakova@dk2n25 ~ $ touch 111.sh
sakudyakova@dk2n25 ~ $ chmod +x 111.sh
sakudyakova@dk2n25 ~ $ bash 111.sh -p улит -i input.txt -o output.txt -c -n
sakudyakova@dk2n25 ~ $
```

Рис. 4.1: Создание файла

Командный файл, с командами `getopts` и `grep`, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-rшаблон` — 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. 4.2).

```

1 #!/bin/bash
2 while getopts i:o:p:cn optletter
3 do
4 case $optletter in
5 i) iflag=1; ival=$OPTARG;;
6 o) oflag=1; oval=$OPTARG;;
7 p) pflag=1; pval=$OPTARG;;
8 c) cflag=1;;
9 n) nflag=1;;
10 *) echo Illegal option $optletter;;
11 esac
12 done
13 if ! test $cflag
14 then
15 cf=-i
16 fi
17 if test $nflag
18 then
19 nf=-n
20 fi
21 grep $cf $nf $pval $ival >> $oval

```

Рис. 4.2: 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
```

```
grep $cf $nf $pval $ival >> $oval
```

Результат работы программы в файле output.txt (рис. fig. 4.3).

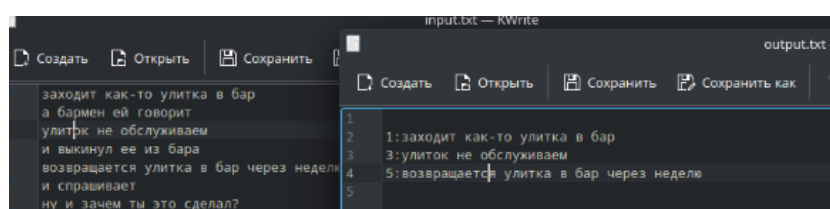


Рис. 4.3: Результат работы программы

Создаю исполняемый файл для второй программы, также создаю файл 12.c для программы на Си (рис. fig. 4.4).

```
sakudyakova@dk2n25 ~ $ touch 112.sh
sakudyakova@dk2n25 ~ $ chmod +x 112.sh
sakudyakova@dk2n25 ~ $ touch 12.cpp
sakudyakova@dk2n25 ~ $ bash 112.sh
sakudyakova@dk2n25 ~ $
```

Рис. 4.4: 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 (рис. fig. 4.5).

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 int main () {
4     int n;
5     printf ("Введите число: ");
6     scanf ("%d", &n);
7     if(n>0){
8         exit(1);
9     }
10    else if (n==0) {
11        exit(0);
12    }
13    else {
14        exit(2);
15    }

```

Рис. 4.5: Код программы на Си

```
#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) {
```

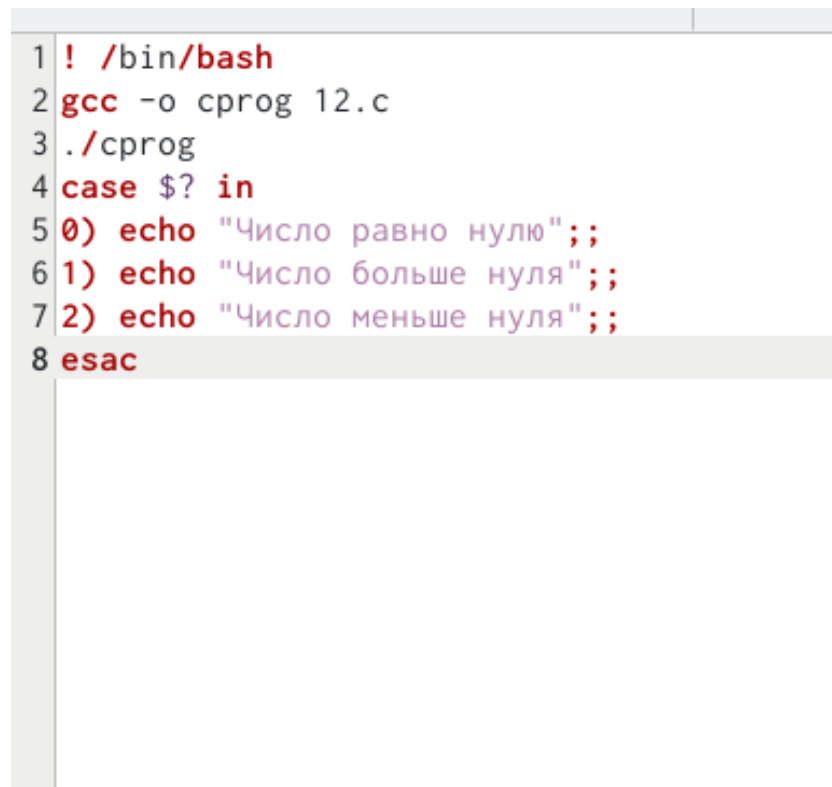
```
        exit(0);
```

```
    }
```

```
    else {
```

```
    exit(2);  
}
```

Командный файл должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено (рис. fig. 4.6).



```
1 #!/bin/bash  
2 gcc -o cprog 12.c  
3 ./cprog  
4 case $? in  
5 0) echo "Число равно нулю";;  
6 1) echo "Число больше нуля";;  
7 2) echo "Число меньше нуля";;  
8 esac
```

Рис. 4.6: Код программы

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

```
gcc -o cprog 12.c
```

```
./cprog
```

```
case $? in
```

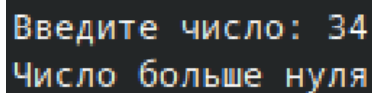
```
0) echo "Число равно нулю";;
```

```
1) echo "Число больше нуля";;
```

```
2) echo "Число меньше нуля";;
```

esac

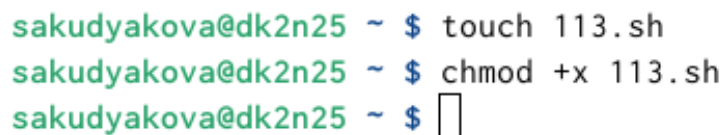
Программа работает корректно (рис. fig. 4.7).



```
Введите число: 34
Число больше нуля
```

Рис. 4.7: Результат работы программы

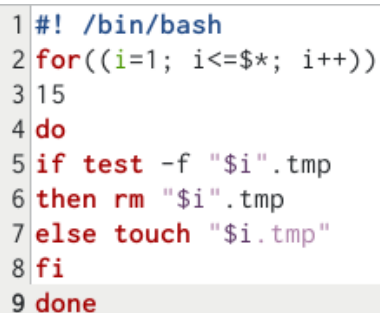
Создаю исполняемый файл для третьей программы (рис. fig. 4.8).



```
sakudyakova@dk2n25 ~ $ touch 113.sh
sakudyakova@dk2n25 ~ $ chmod +x 113.sh
sakudyakova@dk2n25 ~ $
```

Рис. 4.8: 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) (рис. fig. 4.9).



```
1 #!/bin/bash
2 for((i=1; i<=15; i++))
3 do
4   if test -f "$i".tmp
5   then rm "$i".tmp
6   else touch "$i.tmp"
7   fi
8 done
```

Рис. 4.9: 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

```

Проверяю, что программа создала файлы и удалила их при соответствующих запросах (рис. fig. 4.10).

| | | | | | | |
|--------|-------|----------|-----------|------------|-------------|----------------|
| 111.sh | 1.tmp | backup | file1.txt | input.txt | Видео | Музыка |
| 112.sh | 2.tmp | bin | file2.txt | output.txt | Документы | Общедоступные |
| 113.sh | 3.tmp | conf.txt | file.txt | text.txt | Загрузки | 'Рабочий стол' |
| 12.c | 4.tmp | cprog | iloveos | work | Изображения | Шаблоны |

Рис. 4.10: Результат работы программы

Создаю исполняемый файл для четвертой программы. Это 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) (рис. fig. 4.11).

```

1 #!/bin/bash
2 find $* -mtime -7 -mtime +0 -type f > FILES.txt
3 tar -cf archive.tar -T FILES.txt

```

Рис. 4.11: Код программы

```

#!/bin/bash
find $* -mtime -7 -mtime +0 -type f > FILES.txt
tar -cf archive.tar -T FILES.txt

```

Проверяю работу программы (рис. fig. 4.12).

```
sakudyakova@dk2n25 ~ $ bash 114.sh /home/sakudyakova
```

Рис. 4.12: Результат работы программы

5 Выводы

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

6 Ответы на контрольные вопросы

1. Каково предназначение команды getopt?

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 -infile_in.txt -outfile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае:

```
while getopts o:i:Ltr optletter do
case $optletter in
o) iflag=1; ival=$OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option $optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равна `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент. Функция

getopts также понимает переменные типа массив, следовательно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

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`, встречаемая 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.

Список литературы

Архитектура ЭВМ