

Отчёт по лабораторной работе №13

Операционные системы

Бекауов Артур Тимурович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	12
5	Ответы на онтрольные вопросы	13

Список иллюстраций

3.1	Программа №1 - Создание файла	7
3.2	Программа №1 - Выполнение	8
3.3	Программа №2 - создание файлов	8
3.4	Программа №2 - выполнение	10
3.5	Программа №3 - создание файла	10
3.6	Программа №3 - выполнение	11
3.7	Программа №4 - создание файла	11
3.8	Программа №4 - выполнение	11

Список таблиц

1 Цель работы

Цель данной лабораторной работы - 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; `-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 `-r`.
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 `tag` запаковывает в архив все файлы в указанной директории. Модифицировать его так, чтобы запаковывались только те файлы,

3 Выполнение лабораторной работы

Создаю файл lab13-1.sh для новой программы меняю права доступа, разрешая его выполнение, таким образом файл становится исполняемым. (рис. 3.1).

```
[atbekauov@atbekauov ~]$ touch lab13-1.sh
[atbekauov@atbekauov ~]$ chmod +x lab13-1.sh
[atbekauov@atbekauov ~]$ nano lab13-1.sh |
```

Рис. 3.1: Программа №1 - Создание файла

Открываю файл в редакторе nano и записываю следующий 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
```

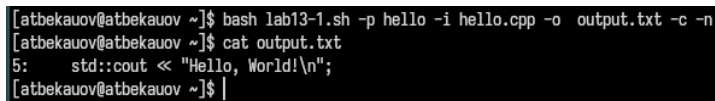
```

if ! test $cflag
    then
        cf=-i
    fi
if test $nflag
    then
        nf=-n
    fi

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

```

Сохраняю файл и закрываю редактор nano, далее запускаю исполняемый файл с помощью команды bash. Затем проверяю, что файл нашёл в программе hello.cpp строочки содержащие hello и вывел их в файл output.txt(рис. 3.2).



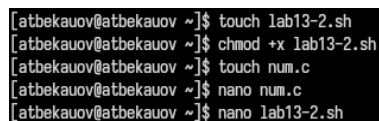
```

[atbekauov@atbekauov ~]$ bash lab13-1.sh -p hello -i hello.cpp -o output.txt -c -n
[atbekauov@atbekauov ~]$ cat output.txt
5:      std::cout << "Hello, World!\n";
[atbekauov@atbekauov ~]$

```

Рис. 3.2: Программа №1 - Выполнение

Создаю файл lab13-2.sh, меняю права доступа, разрешая его выполнение. Также создаю файл num.c - Программу на С. (рис. 3.3).



```

[atbekauov@atbekauov ~]$ touch lab13-2.sh
[atbekauov@atbekauov ~]$ chmod +x lab13-2.sh
[atbekauov@atbekauov ~]$ touch num.c
[atbekauov@atbekauov ~]$ nano num.c
[atbekauov@atbekauov ~]$ nano lab13-2.sh

```

Рис. 3.3: Программа №2 - создание файлов

Открываю файл num.c в nano и ввожу следующий текст программы на С:

```

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

int main ()

```



```

{
    int n;
    printf ("Typein num: ");
    scanf ("%d", &n);
    if (n>0)
    {
        exit(1);
    }
    else if (n==0)
    {
        exit(0);
    }
    else
    {
        exit(2);
    }
}

```

done

Затем открываю в папо файл lab13-2.sh и ввожу следующую программу:

```

#!/bin/bash

gcc -o cprog num.c
./cprog
case $? in
0) echo "Num = 0";;
1) echo "Num > 0";;
2) echo "Num < 0";;
esac

```

Сохраняю файл, выхожу из nano и запускаю файл через bash. Сначала скрипт запускает программу num.c, которая просит на вход число. Затем num.c выходит с соответствующим кодом и скрипт считывает его выводя соответствующее выражение.(рис. 3.4).

```
[atbekauov@atbekauov ~]$ bash lab13-2.sh
Typein num: 1
Num > 0
[atbekauov@atbekauov ~]$ bash lab13-2.sh
Typein num: 0
Num = 0
[atbekauov@atbekauov ~]$ bash lab13-2.sh
Typein num: -1
Num < 0
[atbekauov@atbekauov ~]$
```

Рис. 3.4: Программа №2 - выполнение

Создаю файл lab13-3.sh, меняю права доступа, разрешая его выполнение. Открываю файл в nano (рис. 3.5).

```
[atbekauov@atbekauov ~]$ touch lab13-3.sh
[atbekauov@atbekauov ~]$ chmod +x lab13-3.sh
[atbekauov@atbekauov ~]$ nano lab13-3.sh |
```

Рис. 3.5: Программа №3 - создание файла

Затем ввожу в файл текст программы:

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

Сохраняю файл, выхожу из nano и запускаю файл через bash, скрипт просит ввести кол-во создаваемых файлов - ввожу 4. Скрипт создаёт указанные файлы, а при повторном запуске удаляет их. (рис. 3.6).

```
[atbekauov@atbekauov ~]$ bash lab13-3.sh 4
[atbekauov@atbekauov ~]$ ls
1. tmp      backup      fun          '#lab07-3.sh#'  lab13-3.sh  my_os      prog2.sh  text.txt  Изображения
2. tmp      conf.txt    git-extended '#lab07.sh#'    layout.txt  num.c      prog3.sh  work      Музыка
3. tmp      cprog      hello.cpp    lab07.sh        LICENSE     output.txt prog4.sh  video     Общедоступные
4. tmp      Downloads  '#lab07-1.sh#' lab13-1.sh      may         play       reports   Документы  'Рабочий стол'
australia  feathers   '#lab07-2.sh#' lab13-2.sh      monthly     prog1.sh  ski.places Загрузки  Шаблоны
[atbekauov@atbekauov ~]$ bash lab13-3.sh 4
[atbekauov@atbekauov ~]$ ls
australia  Downloads  hello.cpp    '#lab07.sh#'    lab13-3.sh  monthly    play       prog4.sh  work      Изображения  Шаблоны
backup      feathers   '#lab07-1.sh#' lab07.sh        layout.txt  my_os      prog1.sh  reports   Видео         Музыка
conf.txt    fun        '#lab07-2.sh#' lab13-1.sh      LICENSE     num.c      prog2.sh  ski.places Документы     Общедоступные
cprog       git-extended '#lab07-3.sh#' lab13-2.sh      may         output.txt prog3.sh  text.txt  Загрузки      'Рабочий стол'
```

Рис. 3.6: Программа №3 - выполнение

Создаю файл lab13-4.sh, меняю права доступа, разрешая его выполнение. Открываю файл в nano (рис. 3.7).

```
[atbekauov@atbekauov ~]$ touch lab13-4.sh
[atbekauov@atbekauov ~]$ chmod +x lab13-4.sh
[atbekauov@atbekauov ~]$ nano lab13-4.sh
```

Рис. 3.7: Программа №4 - создание файла

Затем ввожу в файл текст программы:

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

Сохраняю файл, выхожу из nano и запускаю файл через bash, скрипт создал два файла - первый текстовый, который содержит имена архивируемых файлов, второй - архив этих файлов. (рис. 3.8).

```
[atbekauov@atbekauov ~]$ bash lab13-4.sh
[atbekauov@atbekauov ~]$ ls
Arch.tar  cprog      hello.cpp    lab07.sh        layout.txt  num.c      prog3.sh  work      Музыка
Arch.txt  Downloads  '#lab07-1.sh#' lab13-1.sh      LICENSE     output.txt prog4.sh  video     Общедоступные
australia  feathers   '#lab07-2.sh#' lab13-2.sh      may         play       reports   Документы  'Рабочий стол'
backup     fun        '#lab07-3.sh#' lab13-3.sh      monthly     prog1.sh  ski.places Загрузки  Шаблоны
conf.txt   git-extended '#lab07.sh#'    lab13-4.sh      my_os      prog2.sh  text.txt  Изображения
```

Рис. 3.8: Программа №4 - выполнение

4 Выводы

В ходе данной лабораторной работы я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Ответы на онтрольные вопросы

1. Каково предназначение команды getoptс?

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) iflag = 1; oval =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, следова-

тельно, 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`, 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.