

Отчёт по лабораторной работе №11

Дисциплина: Операционные системы

Батова Ирина Сергеевна, НММбд-01-22

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Выводы	17
5	Контрольные вопросы	18

Список иллюстраций

3.1	Создание файла 'file21.sh'	8
3.2	Программа 1	9
3.3	Проверка корректности программы 1	10
3.4	Создание файлов 'file22.c' и 'file22.sh'	10
3.5	Программа 2 на языке Си	11
3.6	Программа 2	11
3.7	Проверка корректности программы 2	12
3.8	Создание файла 'file23.sh'	12
3.9	Программа 3	13
3.10	Проверка корректности программы 3	14
3.11	Создание файла 'file24.sh'	14
3.12	Программа 4	15
3.13	Проверка корректности программы 4	16

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.

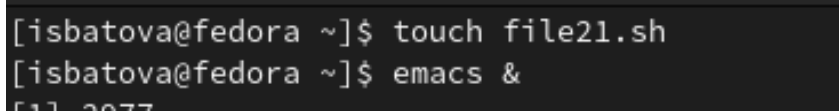
2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
 - `inputfile` — прочитать данные из указанного файла;
 - `outputfile` — вывести данные в указанный файл;
 - `ршаблон` — 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 Выполнение лабораторной работы

1. Для начала работы создаем файл для написания скрипта и открываем его (рис. 3.1).



```
[isbatova@fedora ~]$ touch file21.sh
[isbatova@fedora ~]$ emacs &
[1] 2077
```

Рис. 3.1: Создание файла 'file21.sh'

Нам необходимо, используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами, а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.

Вводим скрипт в наш файл (рис. 3.2).


```
#!/bin/bash

iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:o:p:C:n opt
do case $opt in
    i) iflag=1; ival=$OPTARG;;
    o) oflag=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    C) cflag=1;;
    n) nflag=1;;
    *) echo illegal option $opt
       esac
done

if (($pflag==0))
then echo "Не найден шаблон для поиска"
else
    if (($iflag==0))
    then echo "Не найден файл"
    else
        if (($oflag==0))
        then if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival
                else grep -n $pval $ival
                fi
            else if (($nflag==0))
                then grep -i $pval $ival
                else grep -i -n $pval $ival
                fi
            fi
        else if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival > $oval
                else grep -n $pval $ival > $oval
                fi
            else if (($nflag==0))
                then grep -i $pval $ival > $oval
                else grep -i -n $pval $ival > $oval
                fi
            fi
        fi
    fi
fi
fi
```

Рис. 3.2: Программа 1

В данном скрипте мы сначала вводим соответствующие опциям переменные и присваиваем этим переменным 0. Далее программа просматривает командную строку на наличие опций и присваивает 1 тем опциям (переменным), которые есть в командной строке. После этого мы используем команду if для проверки наличия различных опций. Сначала проверяем есть ли шаблон для поиска (слово, которое ищем), а потом есть ли файл, в котором будет искаться это слово. При невыполнении хотя бы одного из условий программа выводит ошибку. Далее мы перебираем различные сочетания опций и выводим соответствующие строки в файл.

Далее добавляем право на выполнение файла командой 'chmod +x *.sh', создаем файл с текстом (test1.txt) и пустой файл, в который выводятся данные (test2.txt) и выполняем скрипт командой './file21.sh (аргументы)'. Для проверки корректности выполнения просматриваем содержимое файла командой cat (рис. 3.3).

```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file21.sh -i test1.txt -o test2.txt -p Markdown -C -n
[isbatova@fedora ~]$ cat test2.txt
1:Markdown – язык разметки, используемый для создания форматированного текста.
3:Markdown содержит базовые элементы, которые можно найти почти в любом README.md.
4:В целом, Markdown используется для быстрого форматирования статьи для перевода в PDF.
[isbatova@fedora ~]$ cat test1.txt
Markdown – язык разметки, используемый для создания форматированного текста.
Почему так?
Markdown содержит базовые элементы, которые можно найти почти в любом README.md.
В целом, Markdown используется для быстрого форматирования статьи для перевода в PDF.
```

Рис. 3.3: Проверка корректности программы 1

2. Для начала работы создаем файл для написания скрипта на языке Си и файл для написания скрипта на bash. Сначала открываем первый файл (рис. 3.4).

```
[isbatova@fedora ~]$ touch file22.c
[isbatova@fedora ~]$ touch file22.sh
[isbatova@fedora ~]$ emacs &
```

Рис. 3.4: Создание файлов 'file22.c' и 'file22.sh'

Нам необходимо, написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. После этого программа должна завершаться с помощью функции exit(n), передавая информацию в о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав его, выдать сообщение о том, какое число было введено.

Вводим скрипт на языке си в файл 'file22.c' (рис. 3.5).

```

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

int main()
{
    printf("Введите число\n");
    int a;
    scanf ("%b", &a);
    if (a<0) exit (0);
    if (a>0) exit (1);
    if (a==0) exit (2);
    return 0;
}

```

Рис. 3.5: Программа 2 на языке Си

В данном скрипте мы запрашиваем у пользователя число, затем программа считывает его и определяет, число больше нуля, равно нулю или меньше нуля.

Далее открываем файл 'file22.sh' и вводим в него скрипт (рис. 3.6).

```

#!/bin/bash
gcc file22.c -o file22
./file22
code=$?
case $code in
    0) echo "Введенное Вами число меньше 0";;
    1) echo "Введенное Вами число больше 0";;
    2) echo "Введенное Вами число равно 0";;
esac

```

Рис. 3.6: Программа 2

В данном скрипте мы сначала компилируем файл с программой на языке Си в объектный файл. Далее вводим команды запуска этого объектного файла, затем программа анализирует, какое число получилось на выходе. После этого с помощью команды "case" выводим соответствующее сообщение.

Далее добавляем право на выполнение файла командой 'chmod +x *.sh' и выполняем скрипт командой './file22.sh'. Для проверки корректности выполнения я ввела несколько чисел из разных диапазонов (рис. 3.7).

```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file22.sh
Введите число
-2
Введенное Вами число меньше 0
[isbatova@fedora ~]$ ./file22.sh
Введите число
5
Введенное Вами число больше 0
[isbatova@fedora ~]$ ./file22.sh
Введите число
0
Введенное Вами число равно 0
```

Рис. 3.7: Проверка корректности программы 2

3. Для начала работы создаем файл для написания скрипта и открываем его (рис. 3.8).

```
[isbatova@fedora ~]$ touch file23.sh
[isbatova@fedora ~]$ emacs &
[1] 3777
```

Рис. 3.8: Создание файла 'file23.sh'

Нам необходимо написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до ☐, число файлов, которые необходимо создать, передаётся в аргументы командной строки. При этом этот же командный файл должен уметь удалять все созданные им файлы (если они существуют).

Вводим скрипт в наш файл (рис. 3.9).

```
#!/bin/bash
opt=$1;
frt=$2;
nmb=$3;
function Func()
{
    for (( i=1; i<=$nmb; i++ )) do
        file=$(echo $frt | tr '#' "$i")
        if [ $opt == "-c" ]
        then
            touch $file
        elif [ $opt == "-r" ]
        then
            rm -f $file
        fi
    done
}
Func
```

Рис. 3.9: Программа 3

В данном скрипте мы вводим переменные для опций, которые будут введены пользователями - опция -r или -c (remove - удалить или create - создать), формат файла и количество файлов, которые нужно создать. После этого задаем функцию Func, которая будет удалять и создавать нужное количество файлов нужного формата в зависимости от аргументов, и запускаем ее внутри программы.

Далее добавляем право на выполнение файла командой 'chmod +x *.sh' и выполняем скрипт командой './file23.sh (аргументы)'. Для проверки корректности выполнения просматриваем содержимое каталога командой ls и после создания, и после удаления (рис. 3.10).

```
[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ ./file23.sh -c lala#.txt 4
[isbatova@fedora ~]$ ls
backup      file23.sh~  lala4.txt      test2.txt
file1.sh~   file2.sh~   os             work
file21.sh~  file3.sh~   pandoc-3.0     Видео
file21.sh~  file4.sh~   pandoc-3.0-linux-amd64.tar.gz  Документы
file22      forlab11    pandoc-crossref  Загрузки
file22.c    lab07.sh~   pandoc-crossref.1  Изображения
file22.c~   labpractice pandoc-crossref-Linux.tar.xz  Музыка
file22.sh   lala1.txt   stage2          Общедоступные
file22.sh~  lala2.txt   stage3.txt      'Рабочий стол'
file23.sh   lala3.txt   test1.txt       Шаблоны
[isbatova@fedora ~]$ ./file23.sh -r lala#.txt 4
[isbatova@fedora ~]$ ls
backup      file22.sh~  labpractice    stage3.txt  Музыка
file1.sh~   file23.sh~  os             test1.txt   Общедоступные
file21.sh~  file23.sh~  pandoc-3.0     test2.txt   'Рабочий стол'
file21.sh~  file2.sh~   pandoc-3.0-linux-amd64.tar.gz  work        Шаблоны
file22      file3.sh~   pandoc-crossref  Видео
file22.c    file4.sh~   pandoc-crossref.1  Документы
file22.c~   forlab11    pandoc-crossref-Linux.tar.xz  Загрузки
file22.sh   lab07.sh~   stage2          Изображения
```

Рис. 3.10: Проверка корректности программы 3

4. Для начала работы создаем файл для написания скрипта и открываем его (рис. 3.11).

```
[isbatova@fedora ~]$ touch file24.sh
[isbatova@fedora ~]$ emacs &
```

Рис. 3.11: Создание файла 'file24.sh'

Нам необходимо написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории, а затем модифицировать его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад.

Вводим скрипт в наш файл (рис. 3.12).

```
#!/bin/bash
f=$(find ./ -maxdepth 1 -mtime -7)
list=""
for file in "$f" ; do
    file=$(echo "$file" | cut -c 3-)
    list="$list $file"
done
catalog=$(basename $(pwd))
tar -cvf $catalog.tar $list
```

Рис. 3.12: Программа 4

В данном скрипте сначала с помощью команды `find` находим файлы, которые были изменены меньше недели назад и создаем переменную для списка файлов, которые будем архивировать. Далее с помощью цикла `for` анализируем каждый файл по времени создания и в зависимости от этого добавляем его в “список файлов” или нет. После окончания цикла архивируем все файлы, содержащиеся в списке.

Далее добавляем право на выполнение файла командой `'chmod +x *.sh'`, переходим в специально созданный каталог `'forlab11'`, в который помещены файлы различного давности, и выполняем скрипт командой `'sudo ~/file24.sh'`. Для проверки корректности выполнения просматриваем содержимое каталога командой `ls` и видим, что был создан архив файлов (рис. 3.13).

```

[isbatova@fedora ~]$ chmod +x *.sh
[isbatova@fedora ~]$ cd ~/forlab11
[isbatova@fedora forlab11]$ ls
exp.doc  exp.pdf  file1.sh  file2.sh  file3.sh  file4.sh  lab07.sh  stage2  stage3.txt
[isbatova@fedora forlab11]$ sudo ~/file24.sh
file2.sh
file1.sh
exp.pdf
exp.doc
file4.sh
file3.sh
stage3.txt
[isbatova@fedora forlab11]$ ls
exp.doc  file1.sh  file3.sh  forlab11.tar  stage2
exp.pdf  file2.sh  file4.sh  lab07.sh      stage3.txt

```

Рис. 3.13: Проверка корректности программы 4

4 Выводы

В данной лабораторной работе мной были изучены основы программирования в оболочке ОС UNIX. Помимо этого, я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

1. Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных.
2. При перечислении имён файлов текущего каталога используются такие метасимволы, как `"*"` (соответствует произвольной, в том числе и пустой строке), `"?"` (соответствует любому одинарному символу), `"[c1-c2]"` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`.
3. В языке `bash` есть следующие операторы действия: `for` (при каждом следующем выполнении оператора цикла `for` переменная принимает следующее значение из списка значений), `case` (реализует возможность ветвления на произвольное число ветвей), `if` (сначала выполняется последовательность команд (операторов), которую задаёт список команд в строке, содержащей служебное слово `if`, а затем, если последняя выполненная команда из этой последовательности команд возвращает нулевой код завершения (истина), то будет выполнена последовательность команд (операторов), которую задаёт список команд в строке, содержащей служебное слово `then`) и `while` (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`).

4. Для прерывания цикла используются операторы `break` (завершает выполнение цикла) и оператор `continue` (завершает данную итерацию блока операторов).
5. Команда `true` всегда возвращает код завершения, равный нулю, и команда `false`, которая всегда возвращает код завершения, не равный нулю.
6. Строка `if test -f mans/i.s, mans/i.s` и является ли он обычным файлом.
7. При замене в операторе цикла `while` служебного слова `while` на `until` условие, при выполнении которого осуществляется выход из цикла, меняется на противоположное.