

# **Отчет по лабораторной работе №12**

**Дисциплина**

Филиппова Анна Дмитриевна

# Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
3	Выводы	13
4	Контрольные вопросы	14

## Список иллюстраций

2.1	Создаем файл . . . . .	5
2.2	Пишем командный файл . . . . .	6
2.3	Пишем командный файл . . . . .	6
2.4	Пишем командный файл . . . . .	6
2.5	Проверка скрипта . . . . .	7
2.6	Создаем файл . . . . .	7
2.7	Пишем командный файл . . . . .	8
2.8	Пишем командный файл . . . . .	8
2.9	Проверка скрипта . . . . .	8
2.10	Проверка скрипта . . . . .	9
2.11	Создаем файл . . . . .	9
2.12	Пишем командный файл . . . . .	10
2.13	Проверка скрипта . . . . .	10
2.14	Проверка скрипта . . . . .	11
2.15	Создаем файл . . . . .	11
2.16	Пишем командный файл . . . . .	11
2.17	Проверка скрипта . . . . .	12

# 1 Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 2 Выполнение лабораторной работы

1. Создаем файл и пишем соответствующие скрипты. (рис. -fig. 2.1) Используя команды `getopts` `grep`, пишем 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` (рис. -fig. 2.2) (рис. -fig. 2.3) (рис. -fig. 2.4)

```
[adfilippova@adfilippova ~]$ touch f.sh
[adfilippova@adfilippova ~]$ emacs &
[1] 3021
[adfilippova@adfilippova ~]$ █
```

Рис. 2.1: Создаем файл

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0; #Инициализация переменных-флагов и присвоение им 0
while getopts i:o:p:C:n optletter #Анализируем командную строку на наличие опций
do case $optletter in #Если опция присутствует в строке то присваиваем ей 1
    i) iflag=1 ival=$OPTARG;;
    o) oflag=1 oval=$OPTARG;;
    p) pflag=1 pval=$OPTARG;;
    C) Cflag=1;;
    n) nflag=1;;
    *) echo illegal option $optletter
        esac
done
if (($pflag==0)) #Проверка указан ли шаблон для поиска
then echo "Шаблон не найден" #Если шаблон не найден то пишем ошибку
else
    if (($iflag==0)) #Проверка существует ли файл
    then echo "Файл не найден" #Если файл не найден по пишем ошибку
    else
        if (($oflag==0))
        then if (($Cflag==0))
        ll... f sh Ton l19 (Shell-script[sh])
```

Рис. 2.2: Пишем командный файл

```
else
    if (($oflag==0))
    then if (($Cflag==0))
        then if (($nflag==0))
            then grep $pval $ival #Файл найден указан шаблон и существуют о
                else grep -n $pval $ival #Файл найден шаблон указан и существ
        fi
    else if (($nflag==0))
        then grep -i $pval $ival #Файл найден указан шаблон и существуют о
    else grep -i -n $pval $ival #Файл найден указан шаблон и существ
fi
fi
else if (($Cflag==0))
then if (($nflag--0))
then grep $pval $ival > $oval #Файл найден указан шаблон и существуют о
else grep -n $oval $ival > $oval #Файл найден указан шаблон и существуют
```

Рис. 2.3: Пишем командный файл

```

❯ опции C p r i i
❯ вуют опции C n p r i i else grep -i -n $pval $ival #Файл найден указан шаблон и существ
    fi
    fi
    else if (($Cflag==0))
    then if (($nflag--0))
    then grep $pval $ival > $oval #Файл найден указан шаблон и существуют о
❯ опции p i i o
    else grep -n $pval $ival > $oval #Файл найден указан шаблон и существую
❯ т опции n p r i i o
    fi
    else if (($nflag==0))
    then grep -i $pval $ival > $oval #Файл найден указан шаблон и существу
❯ ют опции C p r i i o
    else grep -i -n $pval $ival > $oval #Файл найден указан шаблон и существ
❯ вуют опции C n p r i i o
    fi
    fi
    fi
    fi

```

Рис. 2.4: Пишем командный файл

Проверяем работу написанного скрипта, используя различные опции, предварительно добавив право на исполнение файла (команда «`chmod +x f.sh`») и создав

2 файла, которые необходимы для выполнения программы: 1.txt и 2.txt. Скрипт работает корректно. (рис. -fig. 2.5)

```
[adfilippova@adfilippova ~]$ touch 1.txt 2.txt
[adfilippova@adfilippova ~]$ chmod +x f.sh
[adfilippova@adfilippova ~]$ cat 1.txt
[adfilippova@adfilippova ~]$ cat 1.txt
My name is Ann
My NAME is Kate
My family is very beautiful
[adfilippova@adfilippova ~]$ ./f.sh -i 1.txt -o 2.txt -p name -C -n
./f.sh: line 34: syntax error near unexpected token `then'
./f.sh: line 34: `then if (($nflag--0))'
[adfilippova@adfilippova ~]$ ./f.sh -i 1.txt -o 2.txt -p name -C -n
./f.sh: line 34: syntax error near unexpected token `then'
./f.sh: line 34: `then if (($nflag--0))'
[adfilippova@adfilippova ~]$ ./f.sh -i 1.txt -o 2.txt -p name -C -n
[adfilippova@adfilippova ~]$ cat 2.txt
1:My name is Ann
2:My NAME is Kate
[adfilippova@adfilippova ~]$ ./f.sh -i 1.txt -o 2.txt -p name -n
[adfilippova@adfilippova ~]$ cat 2.txt
My name is Ann
[adfilippova@adfilippova ~]$ ./f.sh -i 1.txt -C -n
Шаблон не найден
[adfilippova@adfilippova ~]$ ./f.sh -o 2.txt -p name -C -n
Файл не найден
[adfilippova@adfilippova ~]$ █
```

Рис. 2.5: Проверка скрипта

2. Создаем файлы g.c и g.sh и пишем соответствующие скрипты. (рис. -fig. 2.6)

Пишем 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. (рис. -fig. 2.7) (рис. -fig. 2.8)

```
[adfilippova@adfilippova ~]$ touch g.c g.sh
[1]+  Done                  emacs
[adfilippova@adfilippova ~]$ emacs &
[1] 5109
[adfilippova@adfilippova ~]$ █
```

Рис. 2.6: Создаем файл

```

#include <stdio.h>
#include <stdlib.h>
int main ()
{
    printf ("Введите число \n");
    int a;
    scanf ("%d", &a);
    if (a<0) exit(0);
    if (a>0) exit(1);
    if (a==0) exit(2);
    return 0;
}

```

Рис. 2.7: Пишем командный файл

```

#!/bin/bash
gcc g.c -o g #Компиляция g.c в объектный файл
./g #Запуск объектного файла g
code=$? #Получение кода завершения программы
case $code in #Выбор кода и его расшифровка
    0) echo "Число меньше 0";;
    1) echo "Число больше 0 ";;
    2) echo "Число равно 0";;
esac

```

Рис. 2.8: Пишем командный файл

Проверяем работу написанных скриптов (команда «./g.sh»), предварительно добавив право на исполнение файла (команда «chmod +x g.sh»). Скрипты работают корректно. (рис. -fig. 2.9) (рис. -fig. 2.10)

```

[adfilippova@adfilippova ~]$ chmod +x g.sh

```

Рис. 2.9: Проверка скрипта



```

[adfilippova@adfilippova ~]$ ./g.sh
Введите число
9
Число больше 0
[adfilippova@adfilippova ~]$ ./g.sh
Введите число
0
Число равно 0
[adfilippova@adfilippova ~]$ ./g.sh
Введите число
-3
Число меньше 0
[adfilippova@adfilippova ~]$ █

```

Рис. 2.10: Проверка скрипта

3. Создаем файл и пишем соответствующие скрипты. (рис. -fig. 2.11) Написала 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. 2.12)

```

[adfilippova@adfilippova ~]$ touch h.sh
[adfilippova@adfilippova ~]$ emacs &
[3] 6131
[1] Done emacs
[2] Done emacs
[adfilippova@adfilippova ~]$ █

```

Рис. 2.11: Создаем файл

```
#!/bin/bash
opt=$1; #Инициализация опции либо -с либо -r
format=$2; #Инициализация формата файла
number=$3; #Инициализация количества файлов
function g () #Функция которая отвечает за удаление и создание файлов
{
    for (( i=1; i<=$number; i++)) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt == "-r" ]
        then
            rm -f $file #Удаление файла
        elif [ $opt == "-c" ]
        then
            touch $file #Создание файла
        fi
    done
}
g
```

Рис. 2.12: Пишем командный файл

Проверила работу написанного скрипта (команда «./h.sh»), предварительно добавив право на исполнение файла (команда «chmod +x h.sh»). Создаем три файла (команда «./h.sh -c et#.txt 3»), удовлетворяющие условию задачи, а потом удаляем их (команда «./h.sh -r et#.txt 3»). (рис. -fig. 2.13) (рис. -fig. 2.14)

```
[adfilippova@adfilippova ~]$ chmod +x h.sh
[adfilippova@adfilippova ~]$ ./h.sh -c et#.txt 3
[adfilippova@adfilippova ~]$ ls
123.cpp          et2.txt          Makefile
1.doc            et3.txt          monthly
1.txt            f1.sh            my_os
2.pdf            feathers         new
2.txt            file1.sh         os-introl
3.doc            file1.sh~        pandoc-2.5
4.pdf            file2.sh         pandoc-2.9.2.1-1-amd64.deb.1
abc1             file2.sh~        pandoc-crossref.1
academic-laboratory-report-template
academic-presentation-markdown-template
add              file.sh          pandoc-crossref-Linux.tar.xz
australia        file.sh~         play
backup           file.txt         reports
backup.sh        f.sh             ski.plases
backup.sh~       f.sh~           text.txt
conf.txt         g               usr
e1.txt           g.c             work
e2.txt           g.c~            Видео
e3.txt           g.sh            Документы
e4.txt           g.sh~           Загрузки
ert1.txt         hello.sh         Изображения
ert2.txt         h.sh            Музыка
                 h.sh~           Общедоступные
                 #lab07.sh#      Рабочий стол
```

Рис. 2.13: Проверка скрипта

```
[adfilippova@adfilippova ~]$ ./h.sh -r et#.txt 3
[adfilippova@adfilippova ~]$ ls
123.cpp          f1.sh           monthly
1.doc            feathers        my_os
1.txt            file1.sh        new
2.pdf            file1.sh~       os-intro1
2.txt            file2.sh        pandoc-2.5
3.doc            file2.sh~       pandoc-2.9.2.1-1-amd64.deb.1
4.pdf            file.sh         pandoc-crossref.1
abc1             file.sh~        pandoc-crossref-Linux.tar.xz
academic-laboratory-report-template
academic-presentation-markdown-template
add              file.txt        play
australia        f.sh           reports
backup           f.sh~          ski.plases
backup.sh        g              text.txt
backup.sh~       g.c            usr
conf.txt         g.c~           work
e1.txt           g.sh           Видео
e2.txt           g.sh~          Документы
e3.txt           hello.sh        Загрузки
e4.txt           h.sh           Изображения
ert1.txt         h.sh~          Музыка
ert2.txt         #lab07.sh#     Общедоступные
ert3.txt         lab07.sh       Рабочий стол
                lab07.sh~      Шаблоны
                Makefile
```

Рис. 2.14: Проверка скрипта

4. Создаем файл и пишем соответствующие скрипты. (рис. -fig. 2.15) Пишем командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). (рис. -fig. 2.16)

```
[3]+  Done                  emacs
[adfilippova@adfilippova ~]$ touch j.sh
[adfilippova@adfilippova ~]$ emacs &
```

Рис. 2.15: Создаем файл

```
#!/bin/bash
files=$(find ./-maxdepth 1 -mtime -7) #Поиск файлов в каталоге в котором находится
пользователь
listing="" #Список файлов для архивирования
for file in "$files"; do
    file=$(echo "$file" | cut -c 3-) #Убираем два первых символа из пути к текущему
каталогу чтобы в архиве не создавался каталог
    listing="$listing $file"
done
dir=$(basename $(pwd)) #Текущее имя каталога
tar -cvf $dir.tar $listing #Архивация файлов
```

Рис. 2.16: Пишем командный файл

Проверяем работу написанного скрипта, предварительно добавив право на исполнение файла (команда «chmod +x j.sh») и создав отдельный каталог 1 с

несколькими файлами. Файлы, измененные более недели назад, заархивированы не были. Скрипт работает корректно. (рис. -fig. 2.17)

```
adfilippova@dk3n59 ~/1 $ chmod +x j.sh
adfilippova@dk3n59 ~/1 $ ./j.sh
1.txt
2.txt
j.sh~
tar: 1.tar: файл является архивом; не сброшен
j.sh
adfilippova@dk3n59 ~/1 $ chmod +x j.sh
adfilippova@dk3n59 ~/1 $ ls -l
итого 3
-rw-r--r-- 1 adfilippova studsci  0 мая 27 14:06 1.txt
-rw-r--r-- 1 adfilippova studsci  0 мая 27 14:06 2.txt
-rw-r--r-- 1 adfilippova studsci 497 сен 18  2020 asdfg.asm
-rwxr-xr-x 1 adfilippova studsci 621 мая 27 14:27 j.sh
-rwxr-xr-x 1 adfilippova studsci 620 мая 27 14:20 j.sh~
-rw-r--r-- 1 adfilippova studsci  0 окт  2  2020 prog2.asm
adfilippova@dk3n59 ~/1 $ ./j.sh
1.txt
2.txt
j.sh~
j.sh
adfilippova@dk3n59 ~/1 $
```

Рис. 2.17: Проверка скрипта

## 3 Выводы

Я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

## 4 Контрольные вопросы

1. Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: `getopts option-string variable [arg ...]` Флаги – это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Строка опций `option-string` – это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за символом, обозначающим этот флаг, должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, то она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента. Функция `getopts` также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.
2. 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 лю-

бому 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. Часто бывает необходимо обеспечить проведение каких-либо действий циклически и управление дальнейшими действиями в зависимости от результатов 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. 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. 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). Примеры бесконечных циклов: `while true do echo hello andy done` `until false do echo hello mike done`
6. Строка `if test -f mans/i.s, mans/i.s` является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. 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.