

Отчёт по лабораторной работе №11

дисциплина: Операционные системы

Коннова Татьяна Алексеевна

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	8
4	Контрольные вопросы	17
5	Выводы	20

Список иллюстраций

3.1	Создание нового файла для скрипта	8
3.2	Написание первого скрипта	9
3.3	Право на выполнение, запуск файла и проверка	10
3.4	Создание двух файлов и открытие emacs	10
3.5	Написание программы на языке Си	11
3.6	Написание командного файла для второго задания	12
3.7	Право на выполнение, запуск файла	13
3.8	Создание третьего файла	13
3.9	Написание третьего скрипта	14
3.10	Право на выполнение, запуск файла	15
3.11	Создание четвёртого файла	15
3.12	Написание четвёртого скрипта	15
3.13	Право на выполнение, запуск файла для каталога Catalog1	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.

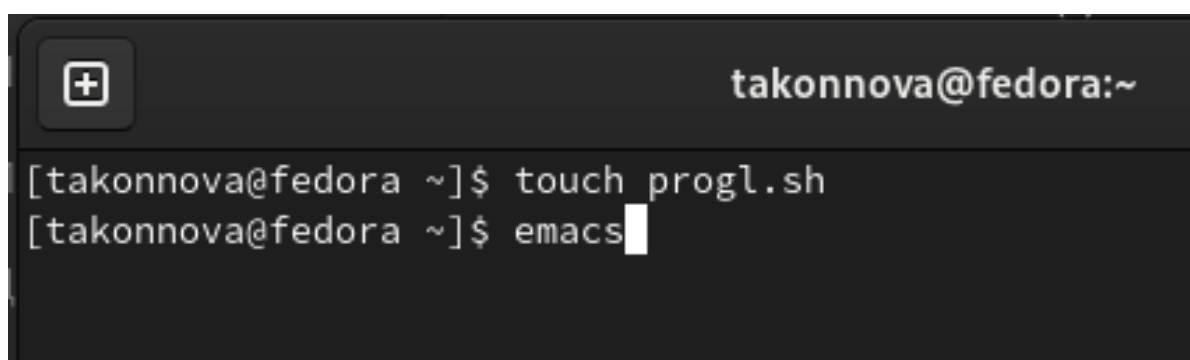
2 Задание

1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами:
 - `-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 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 `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

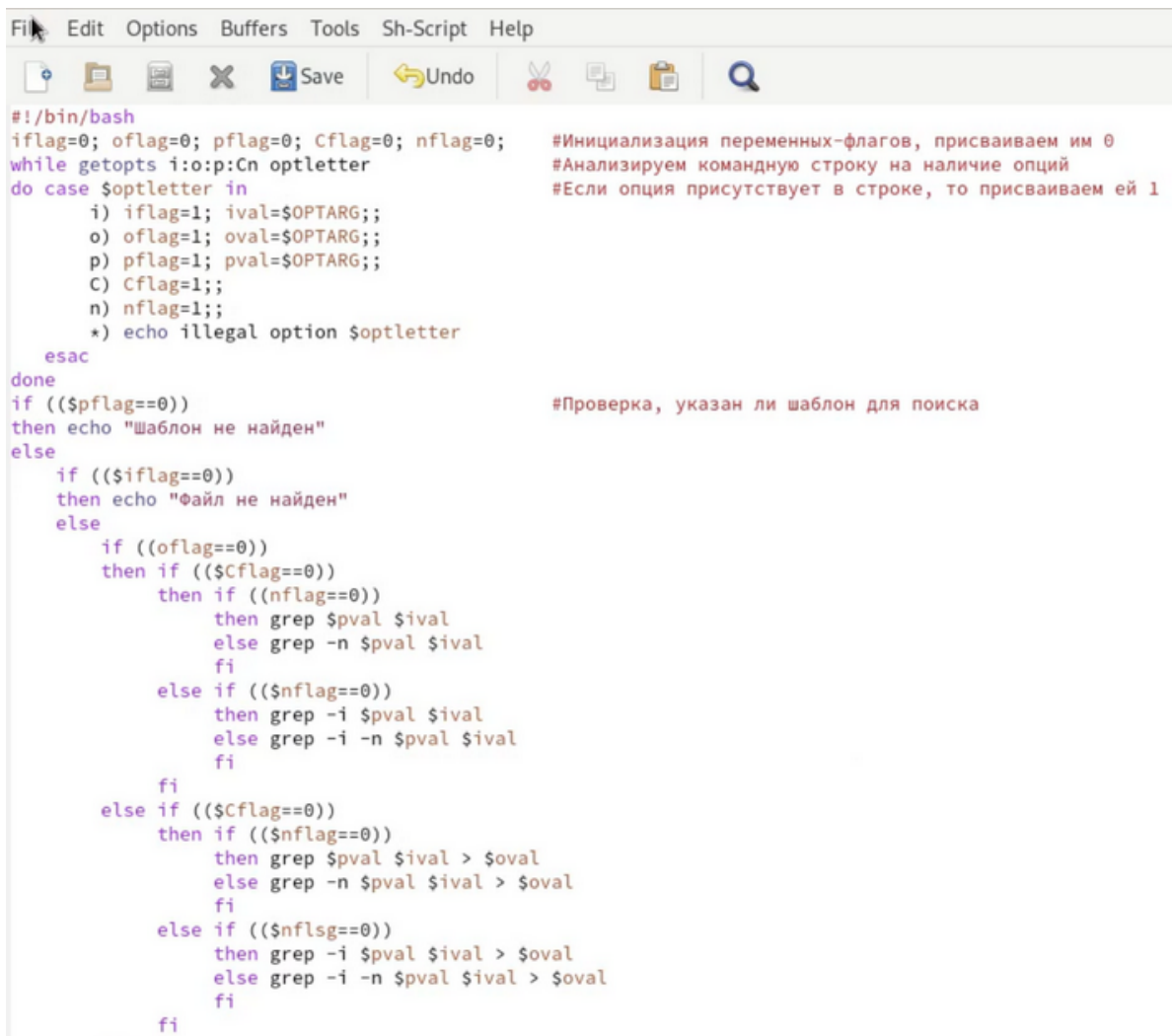
Откроем терминал и создадим в домашнем каталоге файл prog1.sh. После чего перейдём emacs (Рис. [3.1]).

A screenshot of a terminal window with a dark background. The title bar at the top shows a window icon on the left and the text 'takonnova@fedora:~' on the right. The terminal content shows two lines of text: the first line is '[takonnova@fedora ~]\$ touch prog1.sh' and the second line is '[takonnova@fedora ~]\$ emacs' followed by a white cursor block.

```
[takonnova@fedora ~]$ touch prog1.sh
[takonnova@fedora ~]$ emacs
```

Рис. 3.1: Создание нового файла для скрипта

В emacs откроем созданный файл prog1.sh и приступим к написанию скрипта, который анализирует командную строку с определёнными ключами, а затем ищет в указанном файле нужные строки, определяемые ключом p (Рис. [3.2]).



```
#!/bin/bash
iflag=0; oflag=0; pflag=0; Cflag=0; nflag=0; #Инициализация переменных-флагов, присваиваем им 0
while getopts i:o:p:Cn 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"
else
    if (($iflag==0))
    then echo "Файл не найден"
    else
        if (($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
```

Рис. 3.2: Написание первого скрипта

После того как скрипт написан мы сохраняем файл и закрываем emacs. В терминале мы даём этому файлу право на выполнение. Также создаём через терминал два текстовых файла (a1.txt и a2.txt). Далее прописываем нужные нам команды и проверяем корректность работы скрипта (Рис. [3.3]).

```

progl.sh
program.cpp
ski.plases
teers.txt
work
Архитектура
Видео
Документы
Загрузки
Изображения
Музыка
Общедоступные
'Рабочий стол'
Шаблоны
[takonnova@fedora ~]$ mc

[takonnova@fedora ~]$ cat a1.txt
planet you rat[takonnova@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p rat -n
bash: ./progl.sh: Отказано в доступе
[takonnova@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p rat -n
bash: ./progl.sh: Отказано в доступе
[takonnova@fedora ~]$ ./progl.sh -i a1.txt -C -n
bash: ./progl.sh: Отказано в доступе
[takonnova@fedora ~]$

```

Рис. 3.3: Право на выполнение, запуск файла и проверка

Теперь создам в терминале два файла (chslo.c и chslo.sh), для второго задания. Открываем emacs (Рис. [3.4]).

```

[takonnova@fedora ~]$ touch ch.c
[takonnova@fedora ~]$ touch ch.sh
[takonnova@fedora ~]$ emacs

```

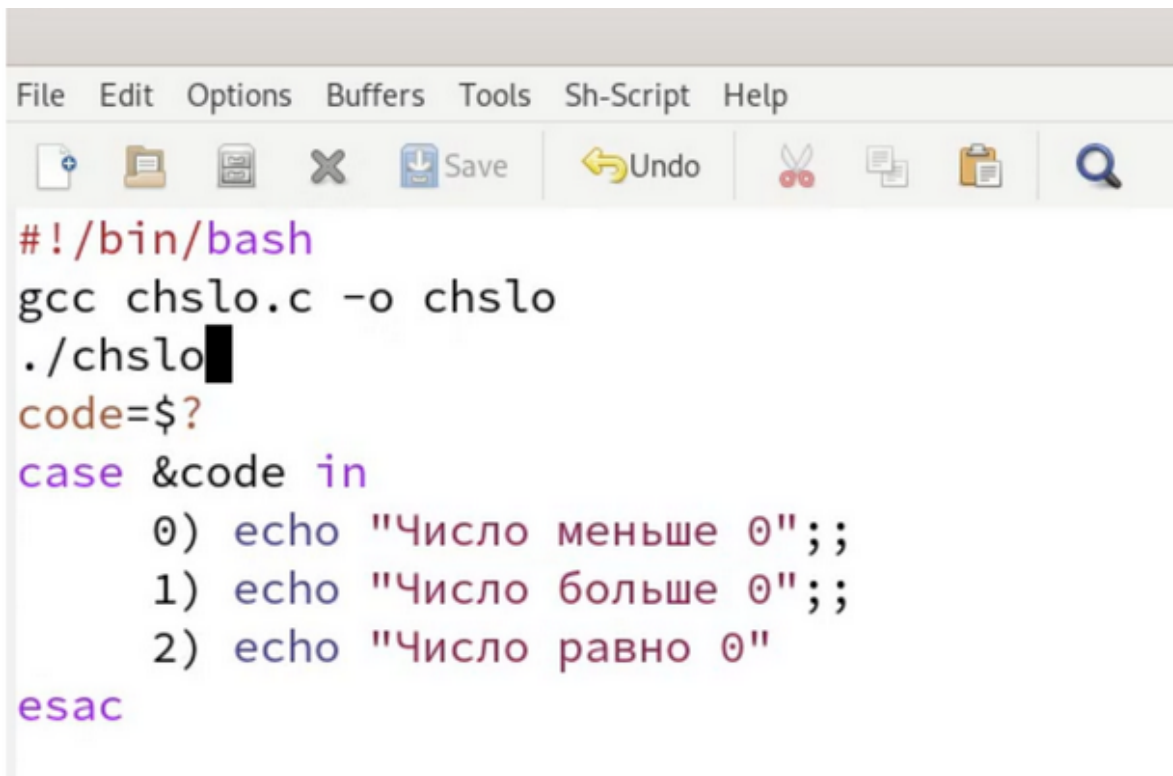
Рис. 3.4: Создание двух файлов и открытие emacs

Открываем файл chslo.c и начинаем писать 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 (Рис. [3.5]).

```
#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. 3.5: Написание программы на языке Си

После программы на языке Си, в файле chslo.sh пишем командный файл, который должен вызывать эту программу и, проанализировав с помощью команды \$?, выдать сообщение о том, какое число было введено (Рис. [3.6]).



The image shows a text editor window with a menu bar (File, Edit, Options, Buffers, Tools, Sh-Script, Help) and a toolbar with icons for file operations and editing. The editor contains a shell script with the following content:

```
#!/bin/bash
gcc chslo.c -o chslo
./chslo
code=$?
case &code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0"
esac
```

Рис. 3.6: Написание командного файла для второго задания

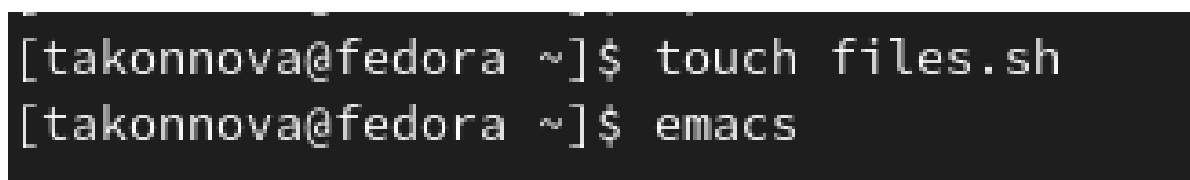
Сохраняем файлы и также даём в терминале право на выполнение для файла chslo.sh. Запускаем файл chslo.sh (Рис. [3.7]).

A terminal window with a dark background. The title bar shows a plus icon on the left and the text 'takonnova@fedora:~' on the right. The terminal content shows a series of commands and their outputs. The user creates 'ch.c' and 'ch.sh' with 'touch'. They attempt to run 'emacs', which fails with a message in Russian. They then run 'chmod +x ch.sh' and './ch.sh', which also fails with the same message. Finally, they run './ch.sh' again, which appears to succeed without an error message.

```
takonnova@fedora:~  
[takonnova@fedora ~]$ touch ch.c  
[takonnova@fedora ~]$ touch ch.sh  
[takonnova@fedora ~]$ emacs  
bash: emacs: команда не найдена...  
^[[АПакеты, предоставляющие этот файл:  
'emacs'  
'emacs-lucid'  
'emacs-nox'  
[takonnova@fedora ~]$ chmod +x ch.sh  
[takonnova@fedora ~]$ ./ch.sh  
[takonnova@fedora ~]$ emacs  
bash: emacs: команда не найдена...  
^[[АПакеты, предоставляющие этот файл:  
'emacs'  
'emacs-lucid'  
'emacs-nox'  
[takonnova@fedora ~]$ ./ch.sh  
[takonnova@fedora ~]$
```

Рис. 3.7: Право на выполнение, запуск файла

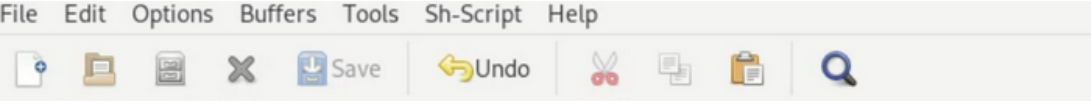
Снова в домашнем каталоге создаём файл, но уже для третьего задания. Запускаем emacs (Рис. [3.8]).

A terminal window with a dark background. The terminal content shows two commands: 'touch files.sh' and 'emacs'.

```
[takonnova@fedora ~]$ touch files.sh  
[takonnova@fedora ~]$ emacs
```

Рис. 3.8: Создание третьего файла

После открытия файла `files.sh` напишем 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. [3.9]).



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

Рис. 3.9: Написание третьего скрипта

Сохраняем наш скрипт и даём право на выполнение. Запускаем файл и создаём три текстовых файла. Удаляем эти три файла (Рис. [3.10]).

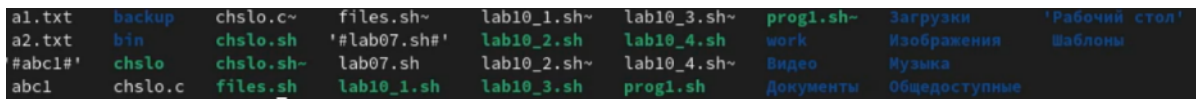


Рис. 3.10: Право на выполнение, запуск файла

Создаём последний файл для четвёртого скрипта. Запускаем emacs (Рис. [3.11]).

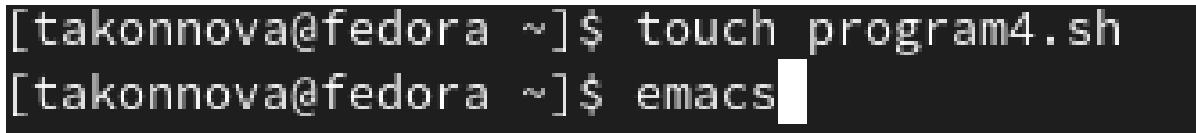


Рис. 3.11: Создание четвёртого файла

В четвёртом файле напишем 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. [3.12]).

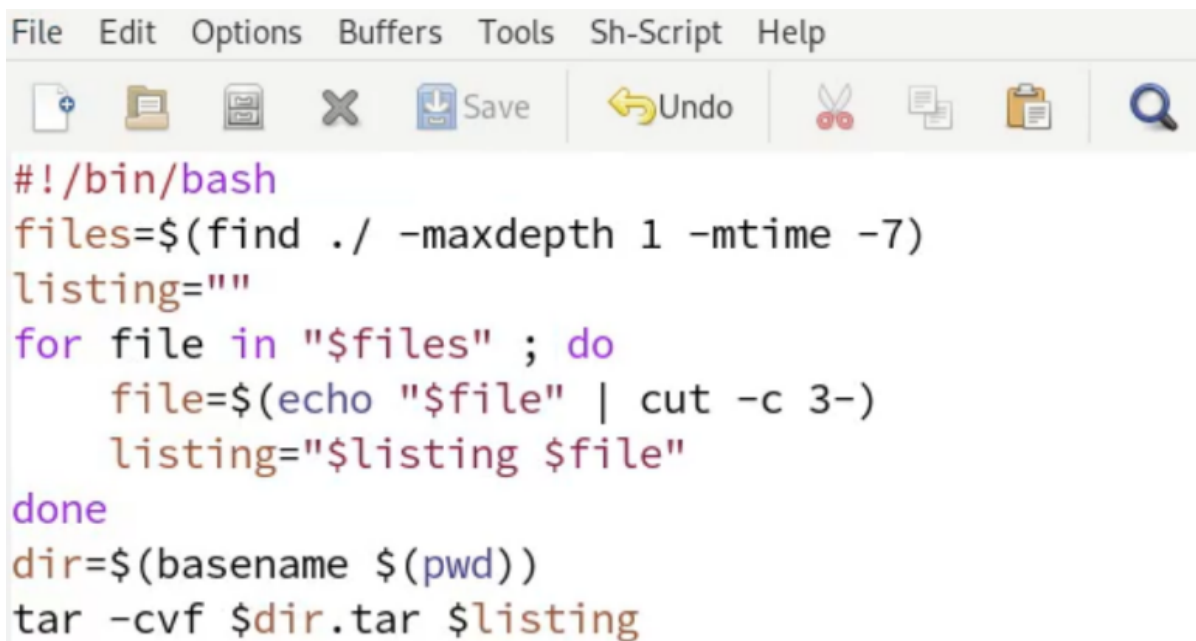
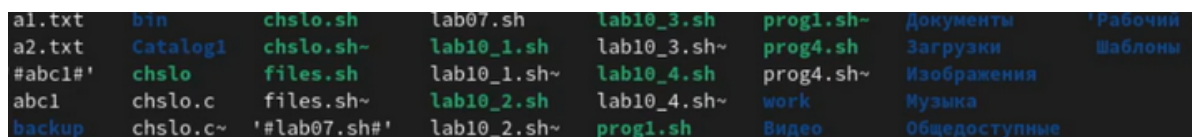


Рис. 3.12: Написание четвёртого скрипта

Сохраним файл и выйдем из emacs. Как делали ранее, дадим файлу право на

выполнение и запустим его для каталога Catalog1 (который мы создали ранее в терминале и в который перенесли некоторые файлы из домашнего каталога) (Рис. [3.13]).



```

a1.txt  bin      chslo.sh  lab07.sh  lab10_3.sh  prog1.sh~  Документы  Рабочий
a2.txt  Catalog1 chslo.sh~ lab10_1.sh lab10_3.sh~ prog4.sh   Загрузки   Шаблоны
#abc1#  chslo    files.sh  lab10_1.sh~ lab10_4.sh  prog4.sh~  Изображения
abc1    chslo.c  files.sh~ lab10_2.sh lab10_4.sh~ work       Музыка
backup  chslo.c~ '#lab07.sh#' lab10_2.sh~ prog1.sh   Видео     Общедоступные
  
```

Рис. 3.13: Право на выполнение, запуск файла для каталога Catalog1

Перейдем в файлы и выполним проверку архивации (Рис. [??]).

Проверка

4 Контрольные вопросы

1. Каково предназначение команды `getopts`?

Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, 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 введённые 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 введённых пользователем данных.

2. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имён файлов текущего каталога можно использовать следующие символы: 1. соответствует произвольной, в том числе и пустой строке; 2. ? соответствует любому одинарному символу; 3. `[c1-c2]` соответствует любому

символу, лексикографически находящемуся между символами `s1` и `s2`. Например, `1.1 echo` выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `1.2. ls.c` выведет все файлы с последними двумя символами, совпадающими с `s.c`. `1.3. echoprogram?` выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.` `1.4.[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 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 `OS/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, неравный нулю (т.е. ложь). Примеры бесконечных циклов: `while true do echo hello andy done` `until false do echo hello mike done`.

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

Строка `if test -f mans/i.s` проверяет, существует ли файл `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.

5 Выводы

В ходе выполнения лабораторной работы мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.