

Лабораторная работа №11

Дисциплина: Операционные системы

Кабанова Варвара Дмитриевна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Ответы на контрольные вопросы	15
4	Выводы	18

List of Figures

2.1	Создание файла	6
2.2	Скрипт №1	7
2.3	Скрипт №1	7
2.4	Скрипт файла a1	8
2.5	Скрипт файла a2	9
2.6	Предоставление прав доступа	9
2.7	Проверка работы программы	10
2.8	Создание файлов	10
2.9	Работа в файле chslo.c	11
2.10	Работа в файле chslo.sh	11
2.11	Проверка скрипта №2	12
2.12	Создание файлов	12
2.13	Скрипт №3	13
2.14	Проверка работы скрипта №3	13
2.15	Создание файлов	14
2.16	Скрипт №4	14

List of Tables

1 Цель работы

Изучение основ программирования в оболочке ОС UNIX. Обучение написанию более сложных командных файлов с использованием логических управляющих конструкций и циклов.

2 Выполнение лабораторной работы

Используя команды `getopts` `grep`, написала командный файл, который анализирует командную строку с ключами: 1. `-i inputfile`—прочитать данные из указанного файла; 2. `-o outputfile`—вывести данные в указанный файл; 3. `-p шаблон` —указать шаблон для поиска; 4. `-C`—различать большие и малые буквы; 5. `-n`—выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.

Для данной задачи я создала файл `prog1.sh` (рис.1) и написала соответствующие скрипты (рис.2-3).

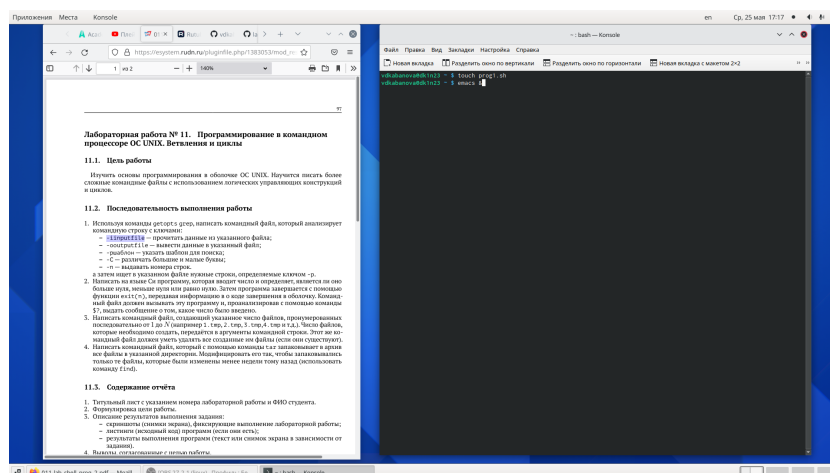


Figure 2.1: Создание файла

```

File Edit Options Buffers Tools Shell-Script Help
emacs@8d51623

#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts :lao:pca:n optletter
do case $optletter in
  l) iflag=1; oval=$OPTARG;;
  o) oflag=1; oval=$OPTARG;;
  p) pflag=1; oval=$OPTARG;;
  c) cflag=1;;
  n) nflag=1;;
  *) echo "Illegal option $optletter"
  esac
done
if (($flag==0))
then echo "Знакомое не найден"
else
if (($flag==0))
then echo "Знаком не найден"
else
if (($flag==0))
then if (($flag==0))
then grep $oval $val
else grep -n $oval $val
fi
else if (($flag==0))
then grep -l $oval $val
else grep -l -n $oval $val
fi
fi
else if (($flag==0))
then if (($flag==0))
then grep $oval $val > $oval
else grep -n $oval $val > $oval
fi
fi
fi
fi
fi

U: 8% - *Warnings* Bot: L8 (Special) Cp: M4M 25 17:32 0.38

```

Figure 2.2: Скрипт №1

```

File Edit Options Buffers Tools Shell-Script Help
emacs@8d51623

else grep -n $oval $val > $oval
fi
else if (($flag==0))
then grep -l $oval $val > $oval
else grep -l -n $oval $val > $oval
fi
fi
fi
fi
fi

U: 8% - *Warnings* Bot: L8 (Special) Cp: M4M 25 17:32 0.38

```

Figure 2.3: Скрипт №1

Проверила работу написанного скрипта, используя различные опции (например, команда «./prog.sh -l a1.txt -o a2.txt -p capital -C n»), предварительно добавив право на исполнение файла (команда «chmod+x prog1.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt (рис.4-7). Скрипт работает корректно.

The screenshot shows an Emacs editor window titled 'emacs@dk1n23'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Text', and 'Help'. The main text area contains the following text:

```
water abc abcs
asd
progl
water water
```

Below the text area, a status bar shows 'U:--- a1.txt All L4 (Text) Ср мая 25 17:36 0.22'. Below this, a warning message is displayed:

```
Warning (initialization): An error occurred while loading '~/.emacs':

error: Package 'fira-code-mode-' is unavailable

To ensure normal operation, you should investigate and remove the
cause of the error in your initialization file. Start Emacs with
the '--debug-init' option to view a complete error backtrace.
```

At the bottom, another status bar shows 'U:%*- *Warnings* All L8 (Special) Ср мая 25 17:36 0.22' and 'Wrote /afs/.dk.sci.pfu.edu.ru/home/v/d/vdkabanova/a1.txt'.

Figure 2.4: Скрипт файла a1

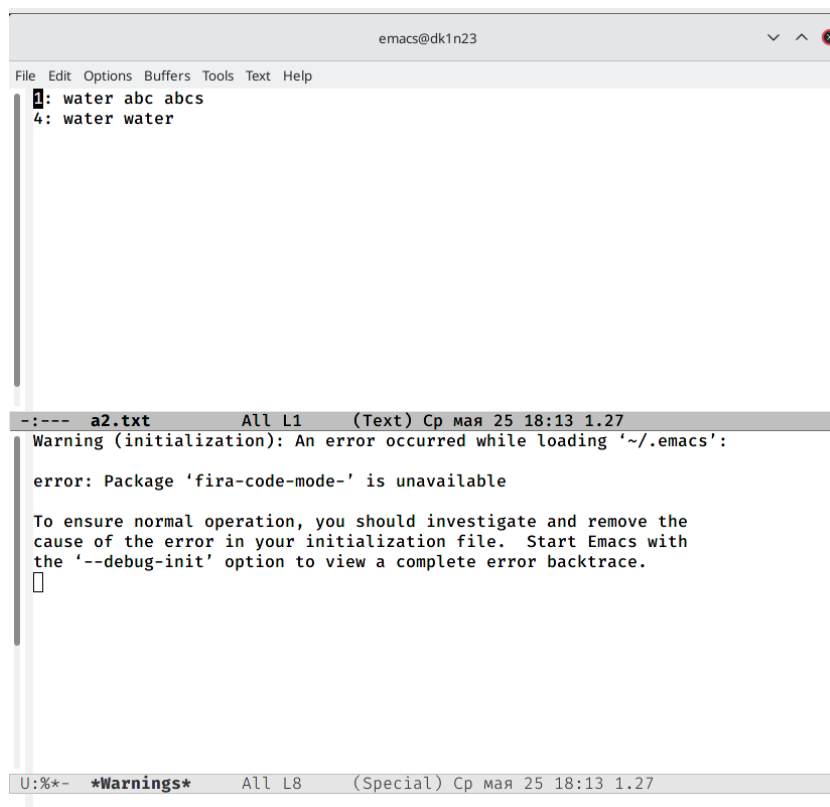


Figure 2.5: Скрипт файла a2

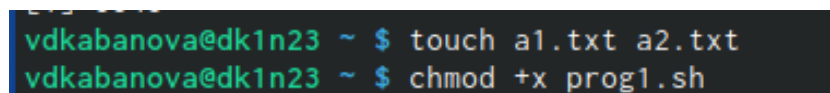


Figure 2.6: Предоставление прав доступа

```

vdkabanova@dk1n23 ~ $ cat a1.txt
water abc abcs
asd
prog1
water water
[2]- Завершён      emacs
[3]+ Завершён      emacs
vdkabanova@dk1n23 ~ $ cat a2.txt
1: water abc abcs
4: water water
vdkabanova@dk1n23 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -n
Шаблон не найден
vdkabanova@dk1n23 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -C -n
Шаблон не найден
vdkabanova@dk1n23 ~ $ ./prog1.sh -i a1.txt -C -n
Шаблон не найден
vdkabanova@dk1n23 ~ $ ./prog1.sh -i a1.txt -o a2.txt -p water -C -n
Шаблон не найден
vdkabanova@dk1n23 ~ $ ./prog1.sh -o a2.txt -p water -C -n
Шаблон не найден

```

Figure 2.7: Проверка работы программы

Написала 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. Для данной задачи я создала 2 файла: `chslo.c` и `chslo.sh` (рис.8) и написала соответствующие скрипты. (команды «`touch prog2.sh`» и «`emacs &`») (рис.9-10).

```

vdkabanova@dk1n23 ~ $ touch chslo.c
vdkabanova@dk1n23 ~ $ touch chslo.sh
vdkabanova@dk1n23 ~ $
vdkabanova@dk1n23 ~ $ emacs &

```

Figure 2.8: Создание файлов

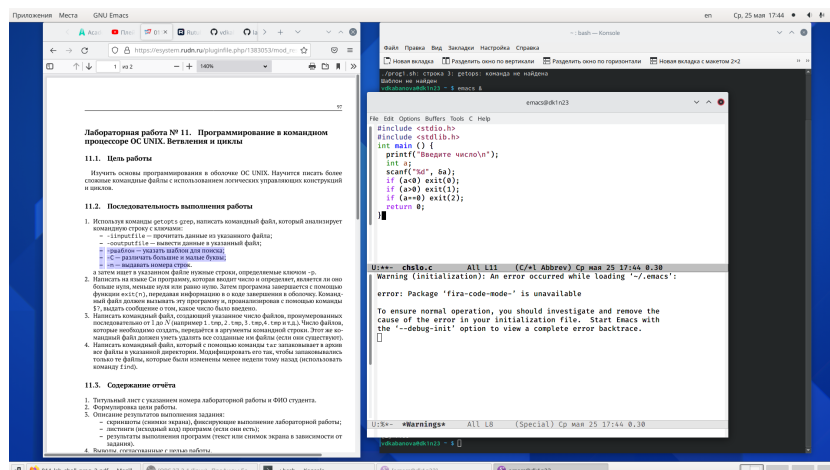


Figure 2.9: Работа в файле chсло.c

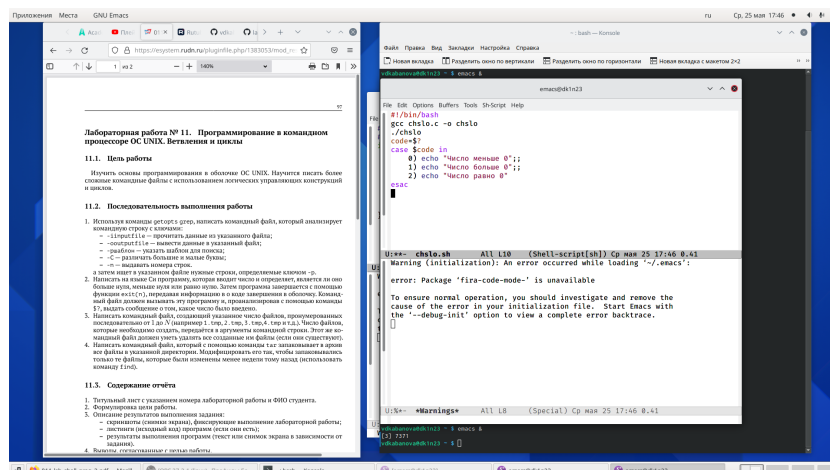


Figure 2.10: Работа в файле chсло.sh

Проверила работу написанных скриптов (команда «./chсло.sh»), предварительно добавив право на исполнение файла (команда «chmod+x chсло.sh») (рис.11). Скрипты работают корректно.

```

vdkabanova@dk1n23 ~ $ chmod +x chslo.sh
vdkabanova@dk1n23 ~ $ ./chslo.sh
Введите число
0
Число равно 0
vdkabanova@dk1n23 ~ $ ./chslo.sh
Введите число
5
Число больше 0
vdkabanova@dk1n23 ~ $ ./chslo.sh
Введите число
-3
Число меньше 0
vdkabanova@dk1n23 ~ $

```

Figure 2.11: Проверка скрипта №2

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). Для данной задачи я создала файл: files.sh (рис.12). и написала соответствующий скрипт (рис.13).

```

vdkabanova@dk1n23 ~ $ touch files.sh
vdkabanova@dk1n23 ~ $ emacs &

```

Figure 2.12: Создание файлов

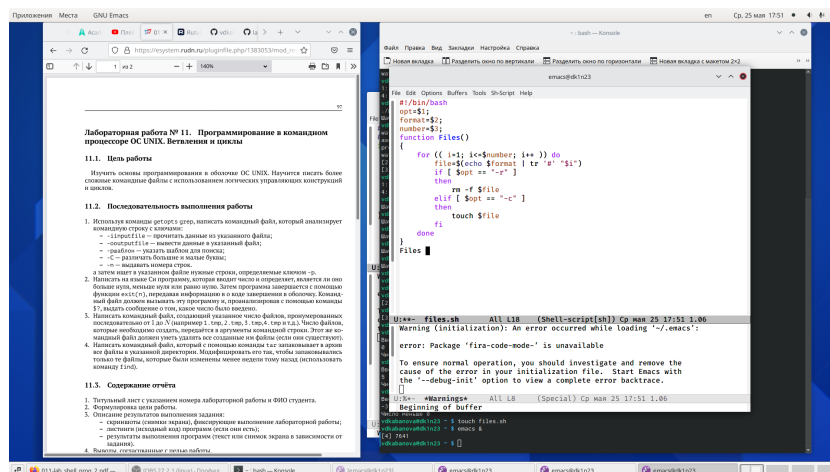


Figure 2.13: Скрипт №3

Далее я проверила работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «chmod+x files.sh»). Сначала я создала три файла (команда «./files.sh-cabc#.txt3»), удовлетворяющие условию задачи, а потом удалила их (команда «./files.sh-rabc#.txt3») (рис.14).

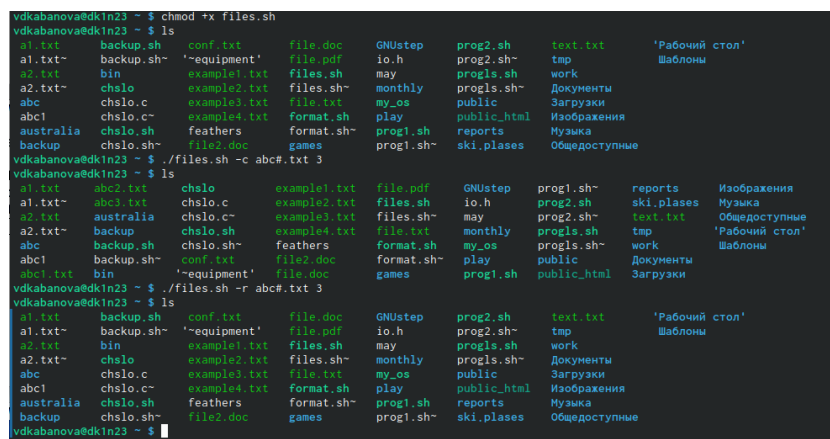


Figure 2.14: Проверка работы скрипта №3

Написала командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицировала его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find). Для данной задачи я создала файл: prog4.sh (рис.15) и написала соответствующий скрипт (рис.16).

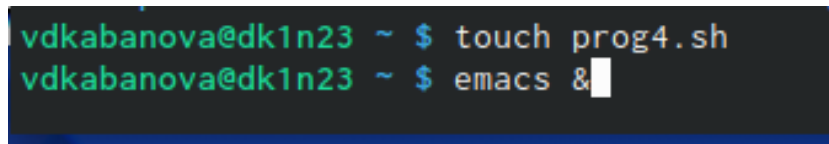


Figure 2.15: Создание файлов

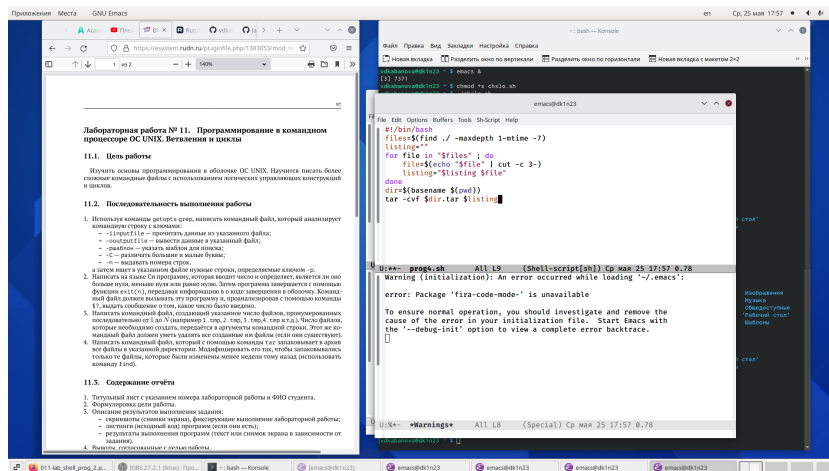


Figure 2.16: Скрипт №4

Далее я проверила работу написанного скрипта (команды «./prog4.sh» и «tar-tf Catalog1.tar»), предварительно добавив право на исполнение файла (команда «chmod +x prog4.sh») и создав отдельный Catalog1 с несколькими файлами. Скрипт работает корректно.

3 Ответы на контрольные вопросы

1. 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 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 любому символу, лексикографически находящемуся между символами c1 и c2. 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.c. 1.3. echoproг.?–выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются прог.. 1.4.[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 иwhile. 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 ОСUNIX 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, неравный 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` и 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.

4 Выводы

В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX/Linux и научилась писать небольшие командные файлы.