

Отчёт по лабораторной работе №11

Дисциплина: Операционные системы

Дупленских Василий Викторович

Содержание

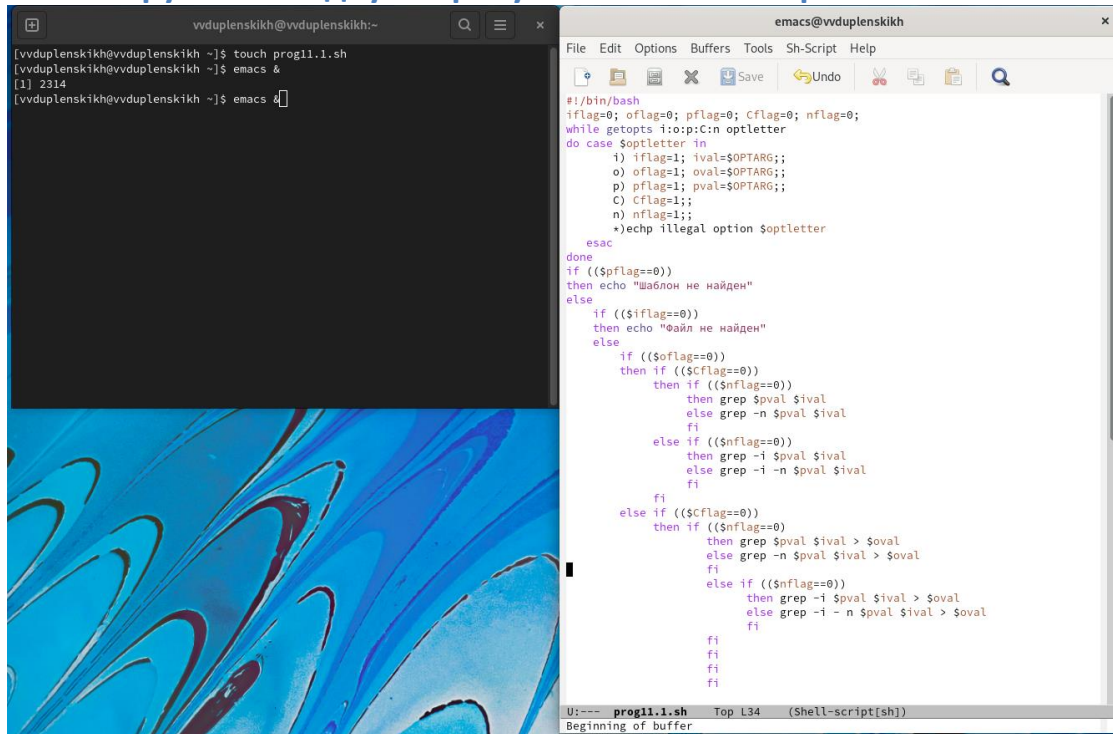
Цель работы:	1
Выполнение лабораторной работы:	2
1.1. Используя команды getopts grep пишу командный файл который анализирует командную строку с ключами -i -o -p -C -n:	2
1.2. Проверяю работу программы, которая читает данные, выводит, читает поисковый шаблон, различает большие и малые буквы и выдает номера строк:.....	3
2.1. Пишу программу на c и в оболочке, которая сравнивает число с 0:.....	3
2.2. Проверяю вторую программу:.....	3
3.1. Пишу программу, которая создает и удаляет заданное количество файлов:	4
3.2. Проверяю третью программу:.....	4
4.1. Пишу программу-архиватор, архивирующая все, что находит в папке, но младше 7 дней:.....	4
4.2. Проверяю четвертую программу:.....	4
Ответы на вопросы:.....	4
Вывод:.....	6

Цель работы:

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

Выполнение лабораторной работы:

1.1. Используя команды `getopts` `grep` пишу командный файл который анализирует командную строку с ключами `-i -o -p -C -n`:



The image shows a terminal window on the left and an Emacs editor window on the right. The terminal window displays the following commands and output:

```
[vvduplenskikh@vvduplenskikh ~]$ touch prog11.1.sh
[vvduplenskikh@vvduplenskikh ~]$ emacs &
[1] 2314
[vvduplenskikh@vvduplenskikh ~]$ emacs &
```

The Emacs editor window shows a shell script named `prog11.1.sh` with the following content:

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:op:C:n 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 $optletter"
  esac
done
if (($pflag==0))
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
```

The status bar at the bottom of the Emacs window shows: `U:---- prog11.1.sh Top L34 (Shell-script[sh]) Beginning of buffer`.

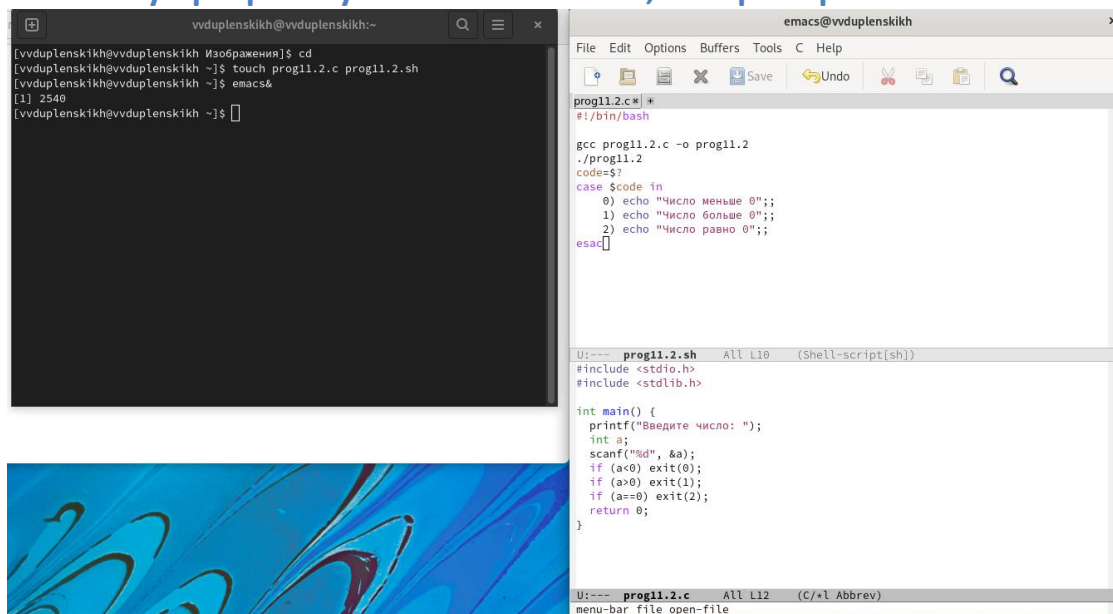
pr1

1.2. Проверяю работу программы, которая читает данные, выводит, читает поисковый шаблон, различает большие и малые буквы и выдает номера строк:

```
[vvduplenskikh@vvduplenskikh ~]$ touch prog11.1.sh
[vvduplenskikh@vvduplenskikh ~]$ emacs &
[1] 2314
[vvduplenskikh@vvduplenskikh ~]$ chmod +x *.sh
[1]+  Завершён      emacs
[vvduplenskikh@vvduplenskikh ~]$ ./prog1.sh -i ~/a.txt -o ~/b.txt -p song -C -n
bash: ./prog1.sh: Нет такого файла или каталога
[vvduplenskikh@vvduplenskikh ~]$ ./prog11.1.sh -i ~/a.txt -o ~/b.txt -p song -C -n
./prog11.1.sh: строка 33: синтаксическая ошибка рядом с неожиданным маркером «else»
./prog11.1.sh: строка 33: `                else grep -n $pval $ival > $oval'
[vvduplenskikh@vvduplenskikh ~]$ ./prog11.1.sh -i ~/a.txt -o ~/b.txt -p song -n
./prog11.1.sh: строка 33: синтаксическая ошибка рядом с неожиданным маркером «else»
./prog11.1.sh: строка 33: `                else grep -n $pval $ival > $oval'
[vvduplenskikh@vvduplenskikh ~]$ ./prog11.1.sh -i ~/a.txt -n
./prog11.1.sh: строка 33: синтаксическая ошибка рядом с неожиданным маркером «else»
./prog11.1.sh: строка 33: `                else grep -n $pval $ival > $oval'
[vvduplenskikh@vvduplenskikh ~]$ ./prog11.1.sh -o ~/b.txt -n
./prog11.1.sh: строка 33: синтаксическая ошибка рядом с неожиданным маркером «else»
./prog11.1.sh: строка 33: `                else grep -n $pval $ival > $oval'
[vvduplenskikh@vvduplenskikh ~]$
```

Программа 1 в действии

2.1. Пишу программу на с и в оболочке, которая сравнивает число с 0:

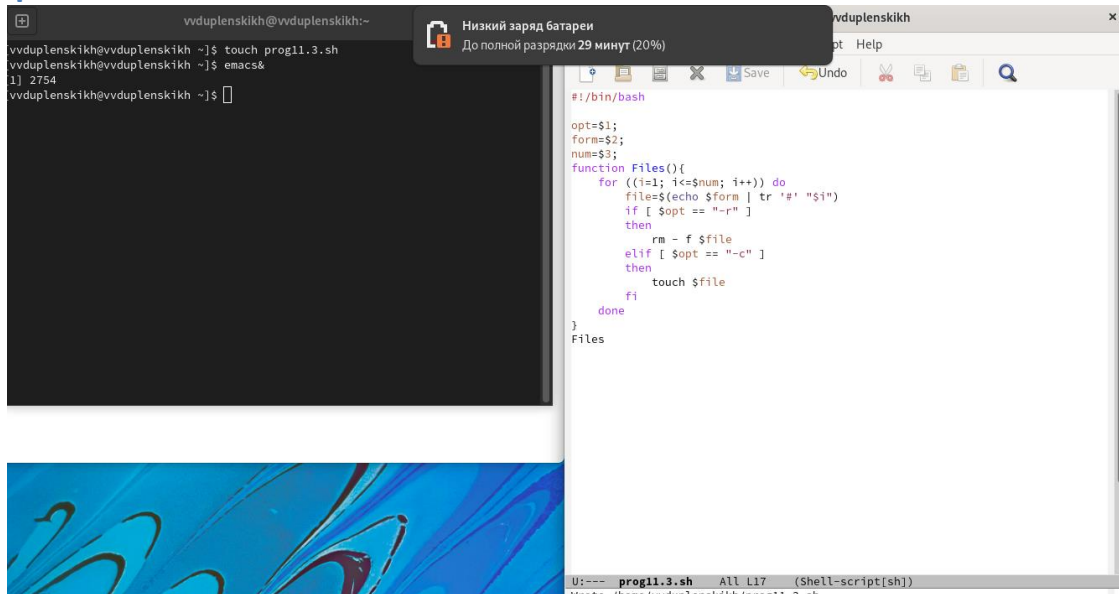


pr2

2.2. Проверяю вторую программу:

![Программа 2 в действии](image/2.2.png)

3.1. Пишу программу, которая создает и удаляет заданное количество файлов:

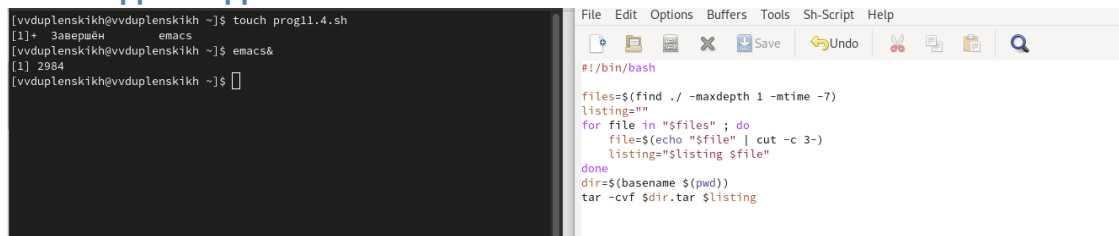


pr3

3.2. Проверяю третью программу:

![Программа 3 в действии(image/3.2.png)]

4.1. Пишу программу-архиватор, архивирующая все, что находит в папке, но младше 7 дней:



pr4

4.2. Проверяю четвертую программу:

![Программа 4 в действии(image/4.2.png)]

Ответы на 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 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. 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. 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` проверяет, существует ли файл `mans/i.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернёт 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.

Еще

Вывод:

Я изучил основы программирования в оболочке ОС UNIX/Linux. Также научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.