

# отчёта по лабораторной работе 12

## Операционные системы

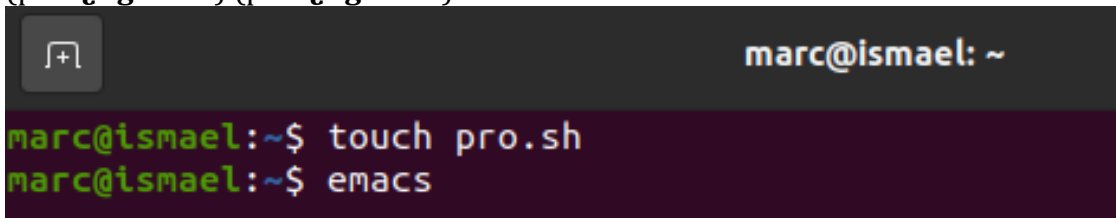
Саинт Амур Измаэль

### Содержание

Цель работы: .....1

### Цель работы:

Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов. ## Ход работы: 1. Использую команды `getopts` `grep`, написала командный файл, который анализирует командную строку с ключами: `-i` `inputfile` — прочитать данные из указанного файла; `-o` `outputfile` — вывести данные в указанный файл; `-r` шаблон — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк, а затем ищет в указанном файле нужные строки, определяемые ключом `-r`. Для данной задачи я создал файл, и открыл его в редакторе `emacs`, использую клавиши «Ctrl-x» и «Ctrl-f» (команды «touch `pro.sh`» и «`emacs`») и написал соответствующие скрипты. (рис. `fig:001?`) (рис. `fig:002?`) (рис. `fig:003?`)



```
marc@ismael: ~  
marc@ismael:~$ touch pro.sh  
marc@ismael:~$ emacs
```

{ # fig:001 width=70%}

```

emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
Save Undo
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:o:p:Cn 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 (($cflag))
then if (($nflag==0))
then grep $pval $ival
else grep -n $pval $ival
fi
else if (($nflag==0))
then grep -i $pval $ival
fi
fi
fi
fi
fi

```

{ # fig:002 width=70%}

```

emacs@ismael
File Edit Options Buffers Tools Sh-Script Help
Save Undo
if (($pflag==0))
then echo "шаблон не найден "
else
if (($iflag==0))
then echo "файл не найден "
else
if (($oflag==0))
then if (($cflag))
then if (($nflag==0))
then grep $pval $ival
else grep -n $pval $ival
fi
else if (($nflag==0))
then grep -i $pval $ival
fi
fi
fi
fi
fi

```

{ # fig:003 width=70%} - Далее я проверил работу написанного скрипта, используя различные опции (например, команда «./pro.sh -l a1.txt -o a2.txt -p capital -C -n»), предварительно добавив право на исполнение файла (команда «chmod +x pro.sh») и создав 2 файла, которые необходимы для выполнения программы: a1.txt и a2.txt.рис. Скрипт работает корректно (рис. 004?) (рис. 005?) (рис. 006?)

```

marc@ismael:~$ touch ab1.txt ab2.txt
marc@ismael:~$ chmod +x pro.sh
marc@ismael:~$ cat ab1.txt
Port-au-Prince is a city in Haiti
Caras is a city in Venezuela
Smolensk is a city in Russia
Miami is a city in USA
marc@ismael:~$

```

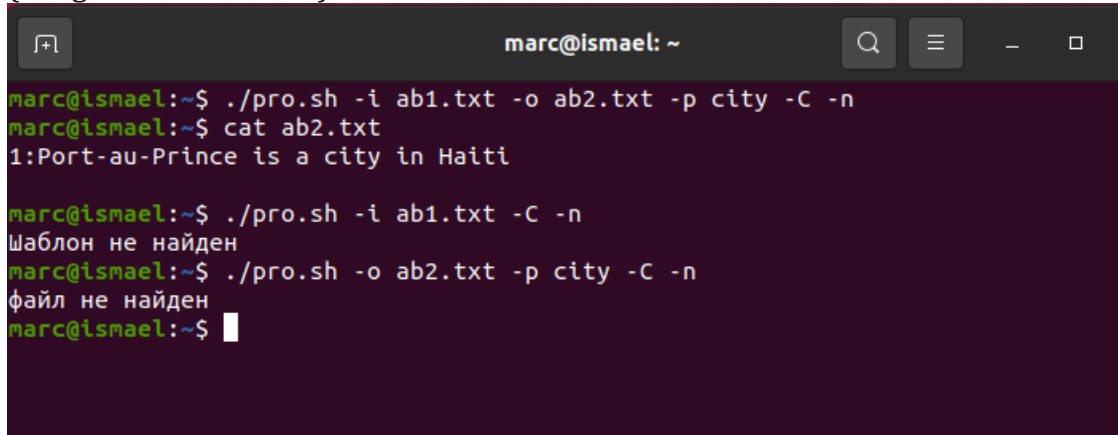
{ # fig:004 width=70%}

```

marc@ismael:~$ ./pro.sh -i ab1.txt -o ab2.txt -p city -C -n
marc@ismael:~$ cat ab2.txt
1:Port-au-Prince is a city in Haiti
2:Caras is a city in Venezuela
3:Smolensk is a city in Russia
4:Miami is a city in USA
marc@ismael:~$

```

{ # fig:005 width=70%}



```

marc@ismael:~$ ./pro.sh -i ab1.txt -o ab2.txt -p city -C -n
marc@ismael:~$ cat ab2.txt
1:Port-au-Prince is a city in Haiti

marc@ismael:~$ ./pro.sh -i ab1.txt -C -n
Шаблон не найден
marc@ismael:~$ ./pro.sh -o ab2.txt -p city -C -n
файл не найден
marc@ismael:~$

```

{ # fig:006 width=70%} 2. Написал 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: `include.c` и `include.sh` и написал соответствующие скрипты. (рис. -@fig:007) с ![7](image/7.png){ # fig:007 width=70%} ![8](image/8.png){ # fig:008 width=70%} - Далее я проверил работу написанных скриптов (команда «./include.sh»), предварительно добавив право на исполнение файла (команда «`chmod +x include.sh`») Скрипты работают корректно. (рис. -@fig:009) ![9](image/9.png){ # fig:009 width=70%}

3. Написал 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 создал файл: `files.sh` и написал соответствующий скрипт. (рис. -@fig:0010) ![10](image/10.png){ # fig:0010 width=70%} - Далее я проверил работу написанного скрипта (команда «./files.sh»), предварительно добавив право на исполнение файла (команда «`chmod +x files.sh`»). Сначала я создал три файла (команда «./files.sh -c abc#.txt 3»), удовлетворяющие условию задачи, а потом удалила их (команда «./files.sh -r abc#.txt 3») (рис. -@fig:0011) (рис. -@fig:0012) (рис. -@fig:0013) ![11](image/11.png){ # fig:0011 width=70%} ![12](image/12.png){ # fig:0012 width=70%} ![13](image/13.png){ # fig:0013 width=70%}

4. Написал командный файл, который с помощью

команды `tar` запаковывает в архив все файлы в указанной директории. Модифицировал 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` и написал соответствующий скрипт. (рис. -@fig:0014) ![14](image/14.png){ # fig:0014 width=70%} - Далее я проверил работу написанного скрипта (команды «`sudo ~/prog4.sh`» и «`tar -tf Catalogue.tar`»), предварительно добавив право на исполнение файла (команда «`chmod +x prog4.sh`») и создав отдельный `Catalogue` с несколькими файлами. Файлы, измененные более недели назад, заархивированы не были. Скрипт работает корректно. (рис. -@fig:0015) (рис. -@fig:0016) (рис. -@fig:0017) ![15](image/15.png){ # fig:0015 width=70%} ![16](image/16.png){ # fig:0016 width=70%} ![17](image/17.png){ # fig:0017 width=70%} \*\*Вывод:\*\* В ходе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX и научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов. \*\*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 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 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 man$/i.s` проверяет, существует ли файл `man$/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.