

# **Отчет по лабораторной работе №11**

**Дисциплина: операционные системы**

Астраханцева А. А.

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>7</b>
<b>5</b>	<b>Выводы</b>	<b>15</b>

## Список иллюстраций

4.1	Создание командного файла №1 . . . . .	7
4.2	Текст программы №1 . . . . .	7
4.3	Проверка работа программы №1 . . . . .	8
4.4	Создание командного файла №2 . . . . .	8
4.5	Текст программы №2 (Си) . . . . .	8
4.6	Текст программы №2 . . . . .	9
4.7	Проверка работа программы №2 . . . . .	9
4.8	Создание командного файла №3 . . . . .	9
4.9	Текст программы №3 . . . . .	10
4.10	Проверка работа программы №3 . . . . .	10
4.11	Создание командного файла №4 . . . . .	10
4.12	Текст программы №4 . . . . .	11
4.13	Проверка работа программы №4 . . . . .	11
4.14	Метасимволы . . . . .	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 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; `-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`.
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  $\infty$  (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 `tag` 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 Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек:

оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux,

C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд;

оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой

BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей со

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ. Стандарты POSIX разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

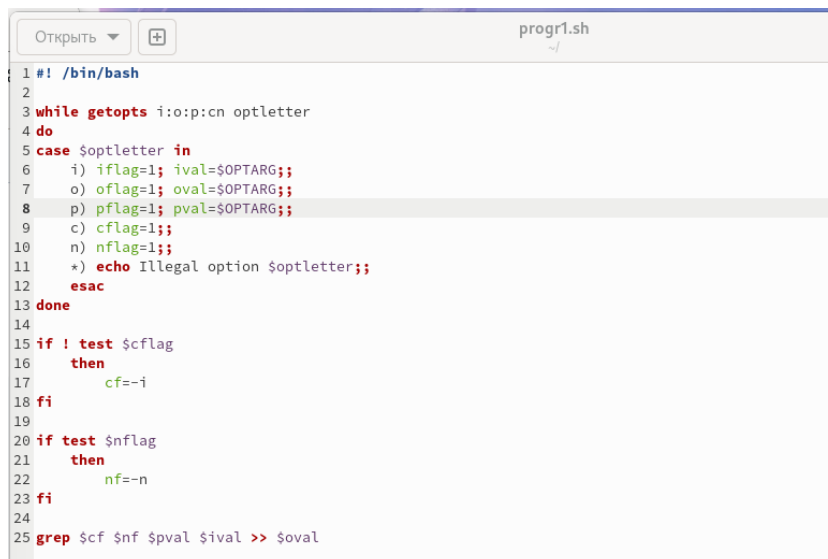
## 4 Выполнение лабораторной работы

Для начала создадим командный файл №1(рис. 4.1).

```
4-linux»: это недопустимый идентификатор
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr1.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup  '#lab07.sh#'  os-intro  work  Изображения  'Рабочий стол'
bin     lab07.sh      progr1.sh  Документы  Музыка        Шаблоны
dir1    lab07.sh~    text.txt  Загрузки  Общедоступные
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.1: Создание командного файла №1

В созданный файл записываем текст нашей программы. Используем оператор case для выбора опций (рис. 4.2).



```
1 #!/bin/bash
2
3 while getopts i:o:p:cn optletter
4 do
5 case $optletter in
6 i) iflag=1; ival=$OPTARG;;
7 o) oflag=1; oval=$OPTARG;;
8 p) pflag=1; pval=$OPTARG;;
9 c) cflag=1;;
10 n) nflag=1;;
11 *) echo Illegal option $optletter;;
12 esac
13 done
14
15 if ! test $cflag
16 then
17     cf=-i
18 fi
19
20 if test $nflag
21 then
22     nf=-n
23 fi
24
25 grep $cf $nf $pval $ival >> $oval
```

Рис. 4.2: Текст программы №1

Проверяем, что работает корректно (рис. 4.3).

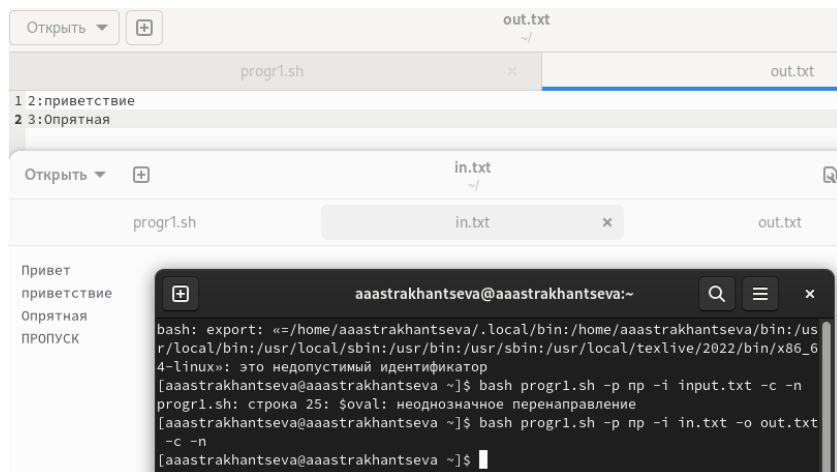


Рис. 4.3: Проверка работа программы №1

Создаем командный файл №2 и файл для программы на Си (рис. 4.4).

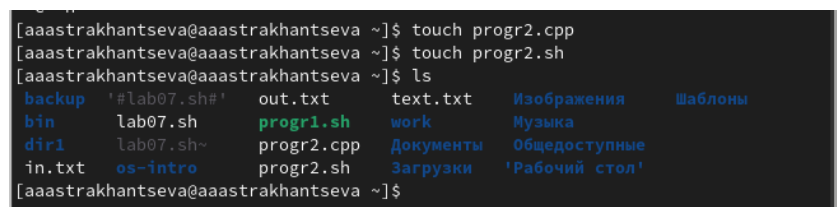


Рис. 4.4: Создание командного файла №2

В созданный файл записываю текст нашей программы на языке Си(рис. 4.5).



Рис. 4.5: Текст программы №2 (Си)



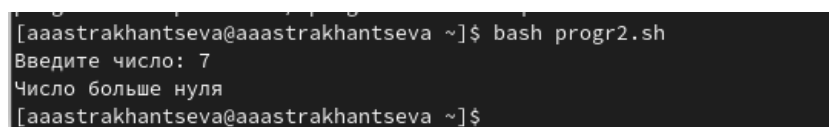
В командный файл записываем текст программы. Используем оператор case (рис. 4.6).



```
1 #!/bin/bash
2 gcc -o cprog progr2.c
3 ./cprog
4 case $? in
5 0) echo "Число равно нулю";;
6 1) echo "Число больше нуля";;
7 2) echo "Число меньше нуля";;
8 esac
```

Рис. 4.6: Текст программы №2

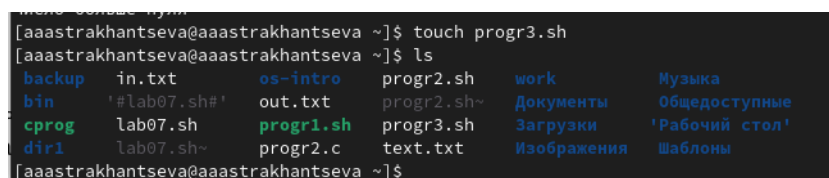
Проверяем, что работает корректно (рис. 4.7).



```
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr2.sh
Введите число: 7
Число больше нуля
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.7: Проверка работа программы №2

Создаем командный файл №3 (рис. 4.8).



```
[aaastrakhantseva@aaastrakhantseva ~]$ touch progr3.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup  in.txt      os-intro    progr2.sh   work        Музыка
bin     '#lab07.sh#' out.txt     progr2.sh~  Документы   Общедоступные
cprog   lab07.sh    progr1.sh   progr3.sh   Загрузки    'Рабочий стол'
dir1    lab07.sh~  progr2.c    text.txt    Изображения Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$
```

Рис. 4.8: Создание командного файла №3

В командный файл записываем текст программы. Используем цикл for (рис. 4.9).

```

Открыть ▾ +
progr2.cpp | progr2.sh |
progr3.sh
~/
#!/bin/bash
for((i=1; i <= $*; i++))
do
if test -f "$i".tmp
then rm "$i".tmp
else touch "$i".tmp
fi
done

```

Рис. 4.9: Текст программы №3

Проверяем, что работает корректно (рис. 4.10).

```

[aaastrakhantseva@aaastrakhantseva ~]$ bash progr3.sh 3
[aaastrakhantseva@aaastrakhantseva ~]$ ls
1.tmp  cprog  lab07.sh~  progr2.sh  Документы  'Рабочий стол'
2.tmp  dir1    os-intro   progr2.sh~  Загрузки   Шаблоны
3.tmp  in.txt  out.txt    progr3.sh  Изображения
backup '#lab07.sh#' progr1.sh  text.txt   Музыка
bin    lab07.sh  progr2.c   work       Общедоступные
[aaastrakhantseva@aaastrakhantseva ~]$ bash progr3.sh 3
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup  in.txt  os-intro  progr2.sh  work  Музыка
bin     '#lab07.sh#' out.txt   progr2.sh~ Документы  Общедоступные
cprog   lab07.sh  progr1.sh progr3.sh  Загрузки  'Рабочий стол'
dir1    lab07.sh~ progr2.c  text.txt   Изображения  Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$

```

Рис. 4.10: Проверка работа программы №3

Создаем командный файл №4 (рис. 4.11).

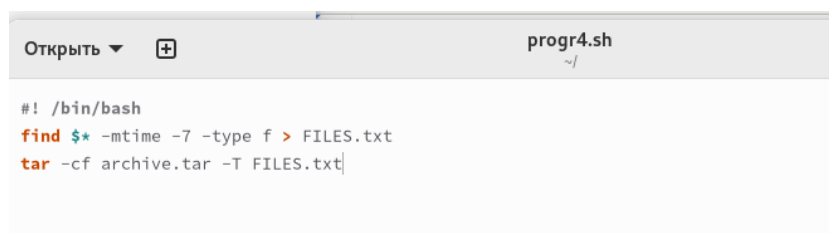
```

[aaastrakhantseva@aaastrakhantseva ~]$ touch progr4.sh
[aaastrakhantseva@aaastrakhantseva ~]$ ды
bash: ды: команда не найдена...
[aaastrakhantseva@aaastrakhantseva ~]$ ls
backup  '#lab07.sh#'  progr1.sh  progr4.sh  Изображения
bin     lab07.sh      progr2.c   text.txt   Музыка
cprog   lab07.sh~    progr2.sh  work       Общедоступные
dir1    os-intro     progr2.sh~ Документы  'Рабочий стол'
in.txt  out.txt      progr3.sh  Загрузки   Шаблоны
[aaastrakhantseva@aaastrakhantseva ~]$

```

Рис. 4.11: Создание командного файла №4

В командный файл записываем текст программы. Используем команды find и tar(рис. 4.12).



```
Открыть ▾ + progr4.sh ~/
#!/bin/bash
find $* -mtime -7 -type f > FILES.txt
tar -cf archive.tar -T FILES.txt|
```

Рис. 4.12: Текст программы №4

Проверяем, что работает корректно (рис. 4.13).

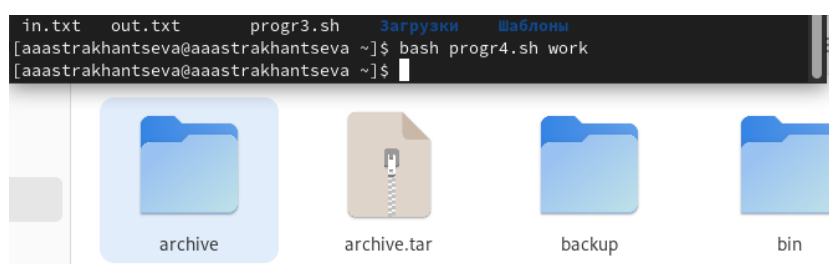


Рис. 4.13: Проверка работа программы №4

## Контрольные вопросы

### 1. Каково предназначение команды getopt?

Команда `getopts` используется shell-процедурами для разбора позиционных параметров и проверки опций на допустимость. Команда `getopts` поддерживает все применимые правила стандартного синтаксиса команд

Цепочка\_опций должна содержать флаги, которые будут распознаваться shell-процедурой, использующей `getopts`. Если за флагом следует двоеточие, то предполагается, что опция требует аргумента.

После каждого вызова `getopts` помещает следующую опцию в переменную имя, а номер аргумента, который должен быть обработан следующим, - в переменную `OPTIND`. Всегда, когда запускается shell или shell-процедура, `OPTIND` получает значение 1.

Если опции требуется аргумент, то `getopts` помещает его в переменную `OPTARG`.

Если встретилась некорректная опция, то в переменную имя помещается ?.

Если опции кончились, getoptс возвращает ненулевой код завершения. Можно использовать специальную опцию –, чтобы отметить конец опций.

По умолчанию команда getoptс разбирает позиционные параметры вызвавшей ее shell-процедуры, но если указать дополнительные аргументы, то getoptс будет разбирать их.

## 2. Какое отношение метасимволы имеют к генерации имён файлов?

При генерации имен используют метасимволы:	
*	произвольная (возможно пустая) последовательность символов;
?	один произвольный символ;
[...]	любой из символов, указанных в скобках перечислением и/или с указанием диапазона;
cat f*	выдаст все файлы каталога, начинающиеся с "f";
cat *f	выдаст все файлы, содержащие "f";
cat program.?	выдаст файлы данного каталога с однобуквенными расширениями, скажем "program.c" и "program.o", но не выдаст "program.com";
cat [a-d]*	выдаст файлы, которые начинаются с "a", "b", "c", "d". Аналогичный эффект дадут и команды "cat [abcd]*" и "cat [bdac]*".

Рис. 4.14: Метасимволы

## 3. Какие операторы управления действиями вы знаете?

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. Какие операторы используются для прерывания цикла?

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, не равный нулю (т.е. ложь).

Что означает строка `if test -f man$s/$i.$s`, встреченная в командном файле?

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).

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

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