

Лабораторная работа №13

Операционные системы

Тойчубекова Асель Нурлановна

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 8 |
| 4 | Выполнение лабораторной работы | 10 |
| 5 | Ответы на вопросы | 17 |
| 6 | Выводы | 19 |

Список иллюстраций

| | | |
|------|--------------------------------|----|
| 4.1 | Создание файла | 10 |
| 4.2 | Редактирование файла | 11 |
| 4.3 | Запуск программы | 11 |
| 4.4 | Вывод программы | 11 |
| 4.5 | Создание файла | 12 |
| 4.6 | Редактирование файла | 12 |
| 4.7 | Редактирование файла | 13 |
| 4.8 | Запуск программы | 13 |
| 4.9 | Создание файла | 14 |
| 4.10 | Редактирование файла | 14 |
| 4.11 | Запуск программы | 15 |
| 4.12 | Новые файлы | 15 |
| 4.13 | Создание файла | 15 |
| 4.14 | Редактирование файла | 15 |
| 4.15 | Запуск программы | 16 |
| 4.16 | Архив файлов | 16 |

Список таблиц

1 Цель работы

Целью данной лабораторной работы является изучить основы программирования в оболочке ОС UNIX. Также научиться 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;
- `-C` — различать большие и малые буквы;
- `-n` — 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 `o` 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 ∞ (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 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, содержащая базовый, но при этом полный набор функций;
- 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 совместимы с операторами оболочки Борна;
- BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

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 Керна.

4 Выполнение лабораторной работы

Для начало я создаю первый файл, в котором буду писать программу и открою его в редакторе gedit. (рис. 4.1).

```
[antoyjchubekova@antoyjchubekova ~]$ touch task11.sh  
[antoyjchubekova@antoyjchubekova ~]$ gedit task11.sh
```

Рис. 4.1: Создание файла

Редактирую файл, пишу командный файл, который используя команды `getopts` `grep` анализирует командную строку с некоторыми ключами. Используя цикл `while` программа анализирует все флаги, запичывая данные в нужные файлы. В конце проверяются использования опций `s` и `p` и присваиваются определенным переменным. Осуществляется команда `grep`, которая в данном случае берет и записывает в новый файл текст который совпал с шаблонным. (рис. 4.2).

```

1 #!/bin/bash
2 while getopts i:o:p:cn optletter
3 do
4 case $optletter in
5 i) i_flag=1; i_file=$OPTARG;;
6 o) o_flag=1; o_file=$OPTARG;;
7 p) p_flag=1; p_file=$OPTARG;;
8 c) c_flag=1;;
9 n) n_flag=1;;
10 *) echo Нет такой опции как $optletter;;
11 esac
12 done
13 if ! test $c_flag
14 then
15 cf=-i
16 fi
17 if test $n_flag
18 then
19 nf=-n
20 fi
21 grep $cf $nf $p_file $i_file >> $o_file
22

```

Рис. 4.2: Редактирование файла

Даю права на выполнение и запускаю программу. (рис. 4.3).

```

[antoychubekova@antoychubekova ~]$ chmod +x task11.sh
[antoychubekova@antoychubekova ~]$ bash task11.sh -p Черный -i output.txt -o input.txt -c -n

```

Рис. 4.3: Запуск программы

Мы видим, что программа удачно сработала и строки с шаблоном “Черный” из текста в файле output.txt записался в файл input.txt. (рис. 4.4).

Рис. 4.4: Вывод программы

Создаю файлы для второй программы и открываю их в редакторе. (рис. 4.5).

```
[antoyjchubekova@antoyjchubekova ~]$ touch task22.sh
[antoyjchubekova@antoyjchubekova ~]$ touch task22.c
[antoyjchubekova@antoyjchubekova ~]$ gedit task22.c
[antoyjchubekova@antoyjchubekova ~]$ gedit task22.sh
```

Рис. 4.5: Создание файла

В файле с расширением .c пишу программу на си, которая вводит число и определяет, является ли оно больше нуля, меньше нуля, или равно нулю. Затем завершающаяся с помощью функции exit(n), передавая информацию о коде завершения в оболочку. Для этого с помощью if проверяю числа и вывожу соответствующий exit. (рис. 4.6).

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 int main (){
5     int n;
6     printf ("Введите число:");
7     scanf ("%d",&n);
8     if(n>0){
9         exit(1);
10    }
11    else if (n==0){
12        exit(0);
13    }
14    else {
15        exit(2);
16    }
17 }
```

Рис. 4.6: Редактирование файла

Открываю файл с расширением .sh и создаю командный файл, который анализирует программу на си с помощью команды \$? и выдает сообщения о том, какое число было введено. Для этого я использую команду gcc -o чтобы программа на си закомпилировалась в новый файл, затем с помощью case проверяю какое значения в этом файле и вывожу соответствующее сообщение. (рис. 4.7).

```
1 #!/bin/bash
2 gcc -o newtask task22.c
3 ./newtask
4 case $? in
5 0) echo "Число=0";;
6 1) echo "Число >0";;
7 2) echo "Число <0";;
8 esac
9
```

Рис. 4.7: Редактирование файла

Даю право на выполнение и запускаю программу с числами 4,0,-8, мы видим, что все работает корректно. (рис. 4.8).

```
[antoychubekova@antoychubekova ~]$ chmod +x task22.sh
[antoychubekova@antoychubekova ~]$ bash task22.sh
Введите число:4
Число >0
[antoychubekova@antoychubekova ~]$ bash task22.sh
Введите число:0
Число=0
[antoychubekova@antoychubekova ~]$ bash task22.sh
Введите число:-8
Число <0
[antoychubekova@antoychubekova ~]$
```

Рис. 4.8: Запуск программы

Создаю третий файл для написания командного файла и открываю его в редакторе. (рис. 4.9).

```
antoyjchubekova@antoyjchubekova ~]$ touch task33.sh
antoyjchubekova@antoyjchubekova ~]$ gedit task33.sh
```

Рис. 4.9: Создание файла

Создаю командный файл, который создает указанное количество файлов, пронумерованные последовательно от 1 до n, число файлов же передается в аргументы командной строки. Также этот же командный файл должен удалять файлы с похожими именами, если они есть до создания новых. Для этого я запрашиваю у пользователя количество файлов, которые необходимо создать и прохожусь от 1 до этого числа параллельно проверяя есть ли уже файлы с такими именами если да, то удаляю их, а если нет создаю их. (рис. 4.10).

```
1 #!/bin/bash
2 echo "Введите число файлов"
3 read n
4 for((i=1;i<=$n;i++))
5 do
6 if test -f "$i".tmp
7 then
8 rm "$i".tmp
9 else touch "$i".tmp
10 fi
11 done
```

Рис. 4.10: Редактирование файла

Даю право на выполнение и запускаю программу с числом 4. (рис. 4.11).

```
[antoyjchubekova@antoyjchubekova ~]$ chmod +x task33.sh
[antoyjchubekova@antoyjchubekova ~]$ bash task33.sh
Введите число файлов
4
[antoyjchubekova@antoyjchubekova ~]$
```

Рис. 4.11: Запуск программы

Зайдя в свой домашний каталог, можно видеть, что файлы удачно созданы. (рис. 4.12).



Рис. 4.12: Новые файлы

Создаю четвертый файл для написания командного файла и открываю его в редакторе. (рис. 4.13).

```
[antoyjchubekova@antoyjchubekova ~]$ touch task44.sh
[antoyjchubekova@antoyjchubekova ~]$ gedit task44.sh
```

Рис. 4.13: Создание файла

Редактирую файл, пишу командный файл, который с помощью команды tar за-паковывает в архив все файлы в указанной директории, также модифицирует его так, чтобы запаковывались только те файлы, которые были изменены менее недели назад. Для этого я использую команду find с опцией -mtime -7(чтобы указать, что менее 7 дней) и -mtime +0(чтобы программа не брала в учет сего-дняшние файлы), а также type -f(для архивации исключительно файлов) вывод этих команд записываю в новый файл 2xfile.txt. Далее архивирую все файлы с помощью команды tar -cf и записываю вывод в файл archive2x.tar. (рис. 4.14).

```
1 #!/bin/bash
2 find * -mtime -7 -mtime +0 -type f > 2xfile.txt
3 tar -cf archive2x.tar -T 2xfile.txt
```

Рис. 4.14: Редактирование файла

Даю право на выполнение и запускаю программу. (рис. 4.15).

```
[antoyjchubekova@antoyjchubekova ~]$ chmod +x task44.sh  
[antoyjchubekova@antoyjchubekova ~]$ bash task44.sh
```

Рис. 4.15: Запуск программы

Перейдя в домашний каталог видим, что программа сработала корректно. (рис. 4.16).



Рис. 4.16: Архив файлов

5 Ответы на вопросы

1. Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных.
2. Метасимволы позволяют использовать шаблоны для сопоставления файлов, основанных на их именах и других атрибутах.
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` заверша-

ет 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).
6. Строка `if test -f mans/i.$s` в командном файле оболочки `Bash` выполняет проверку с помощью утилиты `test` (или `[]`) для определения того, является ли указанный файл обычным файлом.
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.

6 Выводы

В ходе выполнения лабораторной работы № 13 я изучила основы программирования в оболочке ОС UNIX. Также научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов. Создала четыре командных файла и проверила их работу.