

Отчёт по лабораторной работе № 11

Королёв Иван Андреевич

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
4.1	Используя команды <code>getopts</code> <code>grep</code> , написать командный файл, который анализирует командную строку с ключами	8
4.2	Написать на языке Си программу	11
4.3	Написать командный файл, создающий указанное число файлов .	13
4.4	Написать командный файл, который с помощью команды <code>tar</code> упаковывает в архив все файлы в указанной директории	15
5	Выводы	17
6	Контрольные вопросы	18

Список иллюстраций

4.1	Getopts grep	8
4.2	Getopts grep	8
4.3	Getopts grep	9
4.4	Getopts grep	10
4.5	Getopts grep	10
4.6	Си программа	11
4.7	Си программа	11
4.8	Си программа	12
4.9	Си программа	13
4.10	Число файлов	13
4.11	Число файлов	14
4.12	Число файлов	15
4.13	Число файлов	15
4.14	Число файлов	16

Список таблиц

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 ∞ (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, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочке 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 Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

4 Выполнение лабораторной работы

4.1 Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами

Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами (рис. 4.1), (рис. 4.2), (рис. 4.3), (рис. 4.4), (рис. 4.5)

```
[iakorolyov@iakorolyov lab11.1]$ bash file1.sh -p ko -i input.txt -o output.txt -cn  
[iakorolyov@iakorolyov lab11.1]$
```

Рис. 4.1: `Getopts` `grep`



Рис. 4.2: `Getopts` `grep`


```

#!/bin/bash

while getopts i:o:p:cn optletter
do
case $optletter in
    i) iflag=1; ival=$OPTARG;;
    o) oflat=1; oval=$OPTARG;;
    p) pflag=1; pval=$OPTARG;;
    c) cflag=1;;
    n) nflag=1;;
    *) echo Illegal option $optletter
    esac
done

if ! test $cflag
then
    cf=-i
fi

if test $nflag
then
    nf=-n
fi

grep $cf $nf $pval $ival >> $oval

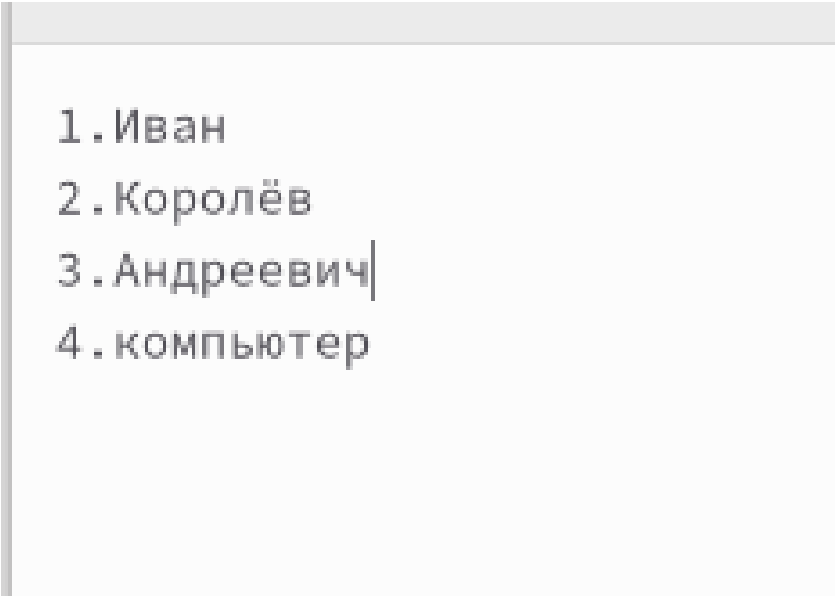
```

Рис. 4.3: Getopts grep



```
4:4.компьютер
```

Рис. 4.4: Getopts grep



```
1.Иван  
2.Королёв  
3.Андреевич  
4.компьютер
```

Рис. 4.5: Getopts grep

4.2 Написать на языке Си программу

Написать на языке Си программу, которая вводит число и определяет, является ли оно больше нуля, меньше нуля или равно нулю. (рис. 4.6), (рис. 4.7), (рис. 4.8), (рис. 4.9)

```
[iakorolyov@iakorolyov lab11.2]$ bash file2.sh
Введите число: 10
Число больше нуля
[iakorolyov@iakorolyov lab11.2]$ bash file2.sh
Введите число: 0
Число равно нулю
[iakorolyov@iakorolyov lab11.2]$ bash file2.sh
Введите число: -1
Число меньше нуля
[iakorolyov@iakorolyov lab11.2]$
```

Рис. 4.6: Си программа

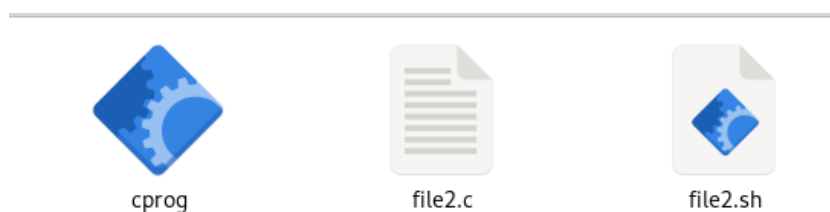


Рис. 4.7: Си программа

```
#include <stdlib.h>
#include <stdio.h>

int main(){
    int n;
    printf ("Введите число: ");
    scanf ("%d", &n);
    if (n>0){
        exit(1);
    } else if (n==0) {
        exit(0);
    } else {
        exit(2);
    }
}
```

Рис. 4.8: Си программа

```
#!/bin/bash

gcc -o cprog file2.c
./cprog
case $? in
0) echo "Число равно нулю";;
2) echo "Число меньше нуля";;
1) echo "Число больше нуля";;
esac
```

Рис. 4.9: Си программа

4.3 Написать командный файл, создающий указанное число файлов

Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (рис. 4.10), (рис. 4.11), (рис. 4.12)

```
[iakorolev@fedora lab11]$ ls
1.tmp 2.tmp 3.tmp 4.tmp 5.tmp 6.tmp file3.sh
[iakorolev@fedora lab11]$
```

Рис. 4.10: Число файлов

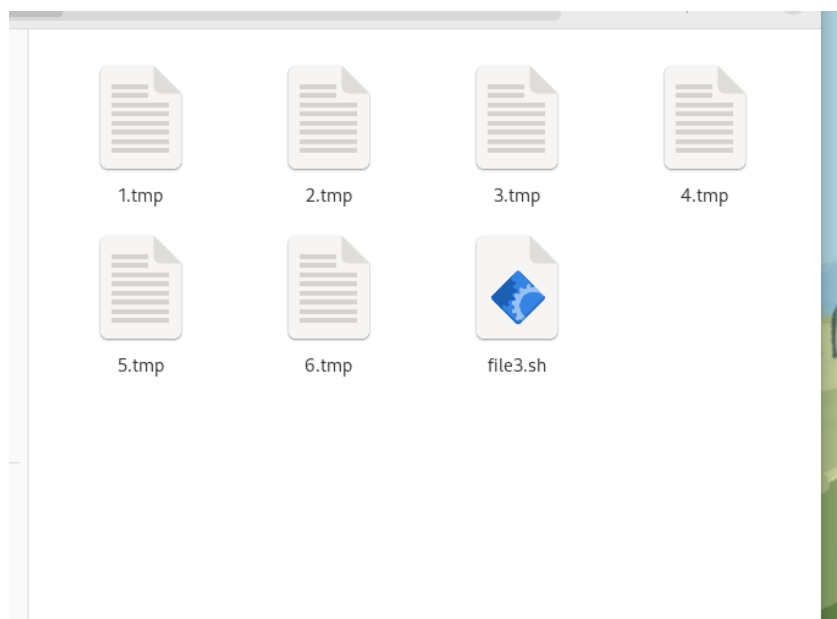


Рис. 4.11: Число файлов

```

1 #!/bin/bash
2 for ((i=1; i<=$*; i++))
3 do
4 if test -f "$i".tmp
5 then rm "$i".tmp
6 else touch "$i".tmp
7 fi
8 done

```

Рис. 4.12: Число файлов

4.4 Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории

Написать командный файл, который с помощью команды tar запаковывает в архив все файлы в указанной директории (рис. 4.13), (рис. 4.14)

```

[iakorolyov@iakorolyov lab11.4]$ bash file4.sh /home/iakorolyov/work
[iakorolyov@iakorolyov lab11.4]$

```

Рис. 4.13: Число файлов

```
1 #!/bin/bash
2
3 find $* -mtime -7 -mtime +6 -type f > FILES.txt
4 tar -cf archive.tar -T FILES.txt
```

Рис. 4.14: Число файлов

5 Выводы

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

6 Контрольные вопросы

1. Каково предназначение команды getopt?

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, `-F` является флагом для команды `ls -F`. Иногда эти флаги имеют аргументы, связанные с ними. Программы интерпретируют эти флаги, соответствующим образом изменяя свое поведение. Строка опций `option-string` — это список возможных букв и чисел соответствующего флага. Если ожидается, что некоторый флаг будет сопровождаться некоторым аргументом, то за этой буквой должно следовать двоеточие. Соответствующей переменной присваивается буква данной опции. Если команда `getopts` может распознать аргумент, она возвращает истину. Принято включать `getopts` в цикл `while` и анализировать введенные данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: `testprog -infile_in.txt -outfile_out.doc -L -t -r` Вот как выглядит использование оператора `getopts` в этом случае:

```
while
getopts o:i:Ltr optletter do
case $optletter in
o) iflag = 1; oval =OPTARG;;
i) iflag=1; ival=$OPTARG;;
L) Lflag=1;;
t) tflag=1;;
r) rflag=1;;
*) echo Illegal option
$optletter
esac
done
```

 Функция `getopts` включает две специальные переменные среды – `OPTARG` и `OPTIND`. Если ожидается дополнительное значение, то `OPTARG` устанавливается в значение этого аргумента (будет равно `file_in.txt` для опции `i` и `file_out.doc` для опции `o`). `OPTIND` является числовым индексом на упомянутый аргумент. Функция `getopts` также понимает переменные типа массив, следова-

тельно, можно использовать ее в функции не только для синтаксического анализа аргументов функций, но и для анализа введенных пользователем данных.

2. Какое отношение метасимволы имеют к генерации имён файлов?

При перечислении имён файлов текущего каталога можно использовать следующие символы: – соответствует произвольной, в том числе и пустой строке; ? – соответствует любому одинарному символу; [с1-с2] – соответствует любому символу, лексикографически находящемуся между символами с1 и с2. Например, `echo *` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .с` – *выведет все файлы с последними двумя символами, совпадающими с .с*. `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. Какие операторы управления действиями вы знаете?

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. Какие операторы используются для прерывания цикла?

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. 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).

6. Что означает строка `if test -f mans/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).

7. 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.