

Лабораторная работа No 11.

Программирование в командном процессоре ОС UNIX. Ветвления и циклы

Сарасбати Брасалес

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	12
6	Контрольные вопросы	13
	Список литературы	16

Список иллюстраций

4.1	8
4.2	8
4.3	9
4.4	9
4.5	9
4.6	10
4.7	10
4.8	10
4.9	11
4.10	11

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; `-rшаблон` — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.
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`).

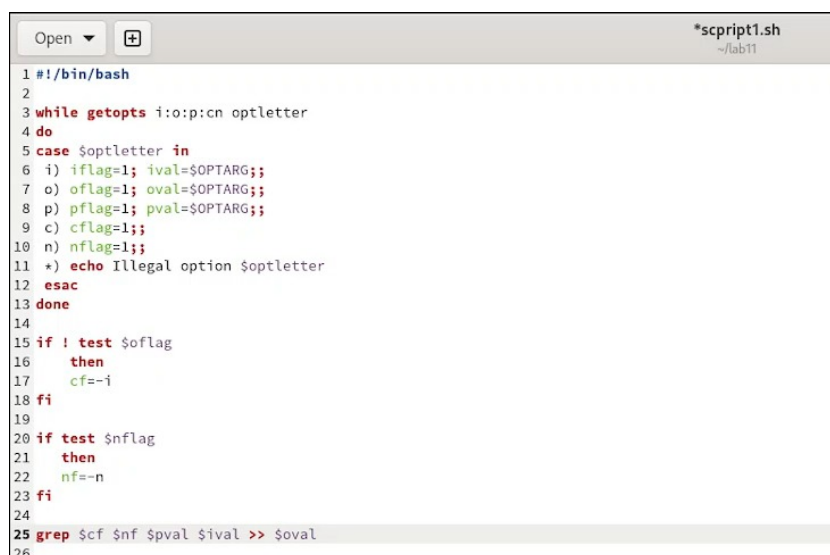
3 Теоретическое введение

Использование команды `getopts` весьма необходимой при программировании является команда `getopts`, которая осуществляет синтаксический анализ командной строки, выделяя флаги, и используется для объявления переменных. Синтаксис команды следующий: 1 `getopts option-string variable [arg ...]` Флаги — это опции командной строки, обычно помеченные знаком минус; Например, для команды `ls` флагом может являться `-F`. Иногда флаги имеют аргументы, связанные с ними. Программы интерпретируют флаги, соответствующим образом изменяя своё поведение. 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 введённые данные с помощью оператора `case`. Предположим, необходимо распознать командную строку следующего формата: 1 `testprog -ifile_in.txt -ofile_out.doc -L -t -r`

4 Выполнение лабораторной работы

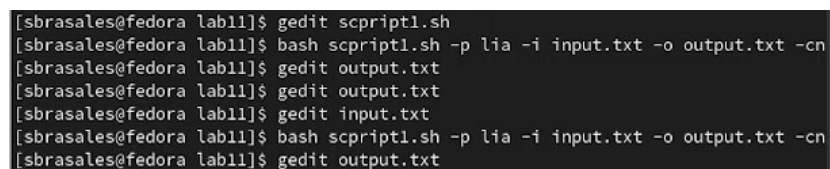
Script 1 (рис. 4.1).

Я создала файл «input.txt» (рис. 4.3) с некоторыми именами, а файл output.txt был пуст, наконец, с помощью команды «bash script..» в файле «output.txt» (рис. 4.4) появляются некоторые имена.



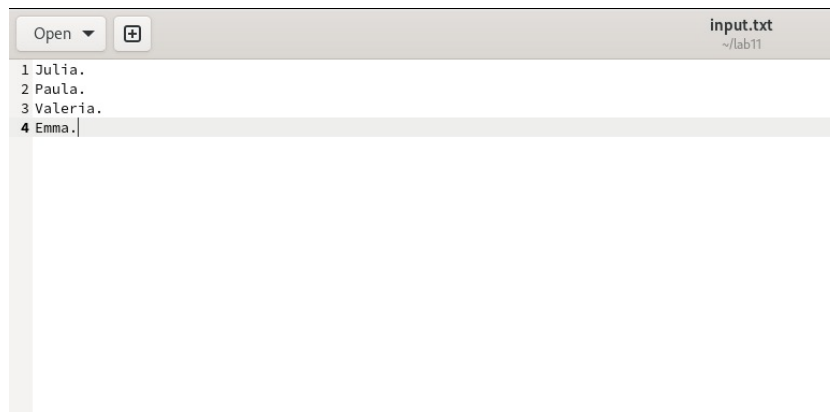
```
1 #!/bin/bash
2
3 while getopts i:o:p:cn optletter
4 do
5 case $optletter in
6 i) iflag=1; ival=$OPTARG;;
7 o) oflag=1; oval=$OPTARG;;
8 p) pflag=1; pval=$OPTARG;;
9 c) cflag=1;;
10 n) nflag=1;;
11 *) echo Illegal option $optletter
12 esac
13 done
14
15 if ! test $oflag
16 then
17 cf=-i
18 fi
19
20 if test $nflag
21 then
22 nf=-n
23 fi
24
25 grep $cf $nf $pval $ival >> $oval
26
```

Рис. 4.1:



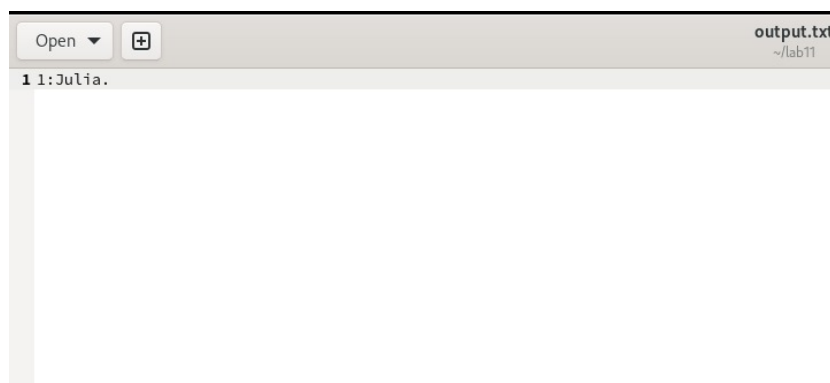
```
[sbrasales@fedora lab11]$ gedit scscript1.sh
[sbrasales@fedora lab11]$ bash scscript1.sh -p lia -i input.txt -o output.txt -cn
[sbrasales@fedora lab11]$ gedit output.txt
[sbrasales@fedora lab11]$ gedit output.txt
[sbrasales@fedora lab11]$ gedit input.txt
[sbrasales@fedora lab11]$ bash scscript1.sh -p lia -i input.txt -o output.txt -cn
[sbrasales@fedora lab11]$ gedit output.txt
```

Рис. 4.2:



```
1 Julia.  
2 Paula.  
3 Valeria.  
4 Emma.
```

Рис. 4.3:



```
1 1:Julia.
```

Рис. 4.4:

Script 2 (рис. 4.5).



```
1 #!/bin/bash  
2  
3 gcc -o cprog program1.c  
4 ./cprog  
5 case $? in  
6 0) echo "Number equal to 0";;  
7 2) echo "Number less than 0";;  
8 3) echo "Number greater than 0";;  
9 esac
```

Рис. 4.5:

программа на C++ (рис. 4.6).



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

Рис. 4.6:

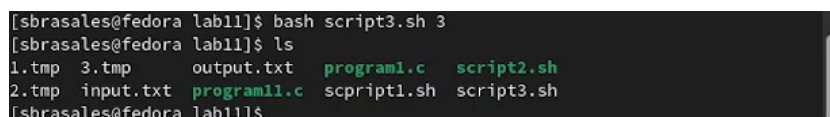
Script 3 (рис. 4.7).



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

Рис. 4.7:

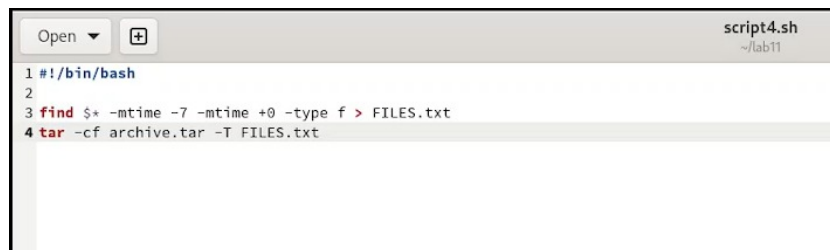
Я создала 3 файла формат tmp (рис. 4.8).



```
[sbrasales@fedora lab11]$ bash script3.sh 3
[sbrasales@fedora lab11]$ ls
1.tmp  3.tmp      output.txt  program1.c  script2.sh
2.tmp  input.txt  program1.c  script1.sh  script3.sh
[sbrasales@fedora lab11]$
```

Рис. 4.8:

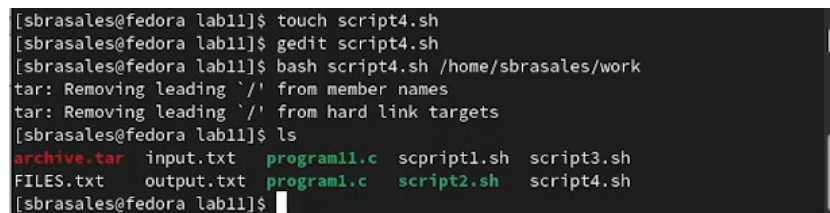
Script 4 (рис. 4.9). (рис. 4.10).



The image shows a text editor window titled "script4.sh" with a path of "~/lab11". The editor contains four lines of code: a shebang line, a blank line, a find command, and a tar command.

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

Рис. 4.9:



The image shows a terminal window with the following commands and output:

```
[sbrasales@fedora lab11]$ touch script4.sh
[sbrasales@fedora lab11]$ gedit script4.sh
[sbrasales@fedora lab11]$ bash script4.sh /home/sbrasales/work
tar: Removing leading '/' from member names
tar: Removing leading '/' from hard link targets
[sbrasales@fedora lab11]$ ls
archive.tar  input.txt  program1.c  scscript1.sh  script3.sh
FILES.txt   output.txt  program1.c  script2.sh   script4.sh
[sbrasales@fedora lab11]$
```

Рис. 4.10:

5 Выводы

Мы изучили основы программирования в оболочке ОС UNIX и научились писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 Контрольные вопросы

Каково предназначение команды `getopts`? Весьма необходимой при программировании является команда `getopts`, которая 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 -ifile_in.txt -ofile_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 равна `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, следовательно, 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 имен файлов текущего каталога можно использовать следующие символы: `·` — соответствует произвольной, в том числе и пустой строке; `· ?` — соответствует любому одному символу; `· [c1-c1]` — соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. `· echo *` — выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `· ls .c` — выведет все файлы с последними двумя символами, равными `.c`. `· echo prog.?` — выдаст 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. Какие операторы управления действиями вы знаете? 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 Вам 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 дела являются операторами языка программирования `bash`. Поэтому при описании языка программирования `bash` термин оператор будет использоваться наравне с термином команда. 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 перестает быть правильным. Пример бесконечного цикла `while`, с прерыванием в момент, когда файл перестает существовать:

`while true do if [! -f file] then break fi sleep 10 done. 5. false true? true : 0. false :`
1.6. *if test* — *fmans/i.s*, встреченная в командном файле? 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. Введенная строка означает условие существования файла *mans/i.s*. 7. Объясните различия между конструкциями *while* и *until*. Если речь идет о 2-х параллельных действиях, то это *while*. когда мы показываем, что сначала делается 1-е действие. потом оно заканчивается при наступлении 2-го действия, применяем *until*.

Список литературы