

Лабораторная работа № 11.

**Программирование в командном процессоре ОС UNIX. Ветвления и
циклы**

ОЗЪЯС Стив Икнэль Дани

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	9
4.1	Задание 1	9
4.2	Задание 2	11
4.3	Задание 3	13
4.4	Задание 4	14
5	Выводы	16
6	Контрольные вопросы	17
	Список литературы	20

Список иллюстраций

4.1	Командный файл №1	10
4.2	Выполнение командного файла №1	10
4.3	Выполнение командного файла №1	11
4.4	Программа на языке Си	12
4.5	Командный файл №2	12
4.6	Выполнение командного файла №2	13
4.7	Командный файл №3	13
4.8	Выполнение командного файла №2	14
4.9	Командный файл №4	14
4.10	Выполнение командного файла №2	15

Список таблиц

1 Цель работы

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;
 - `-` — 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 `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 файл (скрипт или сценарий) - это текстовый файл, состоящий из команд интерпретатора. При запуске этого файла последовательно выполняются все команды, содержащиеся в нем.
- Для создания командного файла:
 - Запустите текстовый редактор.
 - Последовательно запишите команды, располагая каждую команду на отдельной строке.
 - Сохраните этот файл, сделайте его исполняемым, применив команду:
 1. `chmod +x имя_файла.`
 - Запустите созданный файл и проверьте правильность выполнения команд.
 - В случае нахождения ошибки, в текстовом редакторе внесите изменения в командный файл, сохраните его и проверьте еще раз.
- Более подробно об Unix см. в [1–6].

4 Выполнение лабораторной работы

4.1 Задание 1

1. Используя команды `getopts` `grep`, написал командный файл, который анализирует командную строку с ключами:

- `-iinputfile` — прочитать данные из указанного файла;
- `-ooutputfile` — вывести данные в указанный файл;
- `-ршаблон` — указать шаблон для поиска;
- — различать большие и малые буквы;
- `-п` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-р`. (рис. 4.1)

```
sozjyas@fedora:~ -- mcedit script1.sh
script1.sh [----] 0 L: [ 5+31 36/ 37] *(1008/1009b) 0010 0x00A [*]
o) oflag=1 ; outfile=$OPTARG;;
p) pflag=1 ; model=$OPTARG;;
C) cflag=1 ;;
n) nflag=1 ;;
*) echo ERROR $Options
   esac
done

if (( (iflag==1) && (pflag==1) && (oflag==0) ))
then if (( (cflag==0) && (nflag==0) ))
-----> then grep -i -e${model} ${infile}
-----> elif (( (cflag==0) && (nflag==1) ))
-----> then grep -n -i -e${model} ${infile}
-----> elif (( (cflag==1) && (nflag==0) ))
-----> then grep -i -e${model} ${infile}
-----> elif (( (cflag==1) && (nflag==1) ))
-----> then grep -n -e${model} ${infile}
-----> fi
fi

if (( (iflag==1) && (pflag==1) && (oflag==1) ))
then if (( (cflag==0) && (nflag==0) ))
-----> then grep -i -e${model} ${infile} > ${outfile}
-----> elif (( (cflag==0) && (nflag==1) ))
-----> then grep -n -i -e${model} ${infile} > ${outfile}
-----> elif (( (cflag==1) && (nflag==0) ))
-----> then grep -i -e${model} ${infile} > ${outfile}
-----> elif (( (cflag==1) && (nflag==1) ))
-----> then grep -n -e${model} ${infile} > ${outfile}
-----> fi
fi
```

Рис. 4.1: Командный файл №1

- Выполнил его. (рис. 4.2)

```
sozjyas@fedora:~
[sozjyas@fedora ~]$ ./script1.sh -p elif -i script1.sh
  elif (( (cflag==0) && (nflag==1) ))
  elif (( (cflag==1) && (nflag==0) ))
  elif (( (cflag==1) && (nflag==1) ))
  elif (( (cflag==0) && (nflag==1) ))
  elif (( (cflag==1) && (nflag==0) ))
  elif (( (cflag==1) && (nflag==1) ))
[sozjyas@fedora ~]$ ./script1.sh -n -p elif -i script1.sh
16:  elif (( (cflag==0) && (nflag==1) ))
18:  elif (( (cflag==1) && (nflag==0) ))
20:  elif (( (cflag==1) && (nflag==1) ))
28:  elif (( (cflag==0) && (nflag==1) ))
30:  elif (( (cflag==1) && (nflag==0) ))
32:  elif (( (cflag==1) && (nflag==1) ))
[sozjyas@fedora ~]$ ./script1.sh -n -p E -i script1.sh
2:while getopts i:o:p:Cn Options
3:do case $Options in
4:i) iflag=1 ; infile=$OPTARG;;
5:o) oflag=1 ; outfile=$OPTARG;;
6:p) pflag=1 ; model=$OPTARG;;
9:*) echo ERROR $Options
10:   esac
11:done
14:  then if (( (cflag==0) && (nflag==0) ))
15:  then grep -i -e${model} ${infile}
16:  elif (( (cflag==0) && (nflag==1) ))
17:  then grep -n -i -e${model} ${infile}
18:  elif (( (cflag==1) && (nflag==0) ))
```

Рис. 4.2: Выполнение командного файла №1

- Часть 2 (рис. 4.3)

```
sozjyas@fedora:~  
[sozjyas@fedora ~]$ ./script1.sh -Cn -p E -i script1.sh  
9:*) echo ERROR $Options  
[sozjyas@fedora ~]$ ./script1.sh -p elif -i script1.sh -o out1.txt  
[sozjyas@fedora ~]$ ls  
avatar.jpg  Documents      'install-tl-unx (1)'  
backup      Images         'install-tl-unx.tar.gz'  
bin         _index.md     logfile  
Bureau     'install-tl-unx'  Modules  
morefun    out1.txt      Téléchargements  
Musique   Pictures     Vidéos  
newdir    Public       work  
newdir1   script1.sh   z  
[sozjyas@fedora ~]$ cat out1.txt  
elif (( (cflag==0) && (nflag==1) ))  
elif (( (cflag==1) && (nflag==0) ))  
elif (( (cflag==1) && (nflag==1) ))  
elif (( (cflag==0) && (nflag==1) ))  
elif (( (cflag==1) && (nflag==0) ))  
elif (( (cflag==1) && (nflag==1) ))  
[sozjyas@fedora ~]$ ./script1.sh -n -p elif -i script1.sh -o out2.txt  
[sozjyas@fedora ~]$ cat out2.txt  
16: elif (( (cflag==0) && (nflag==1) ))  
18: elif (( (cflag==1) && (nflag==0) ))  
20: elif (( (cflag==1) && (nflag==1) ))  
28: elif (( (cflag==0) && (nflag==1) ))  
30: elif (( (cflag==1) && (nflag==0) ))  
32: elif (( (cflag==1) && (nflag==1) ))  
[sozjyas@fedora ~]$ ./script1.sh -n -p E -i script1.sh -o out2.txt  
[sozjyas@fedora ~]$ ./script1.sh -n -p E -i script1.sh -o out3.txt  
[sozjyas@fedora ~]$ cat out3.txt  
2:while getopts i:op:Cn Options  
3:do case $Options in  
4:i) iflag=1 ; infile=$OPTARG;;  
5:o) oflag=1 ; outfile=$OPTARG;;  
6:p) pflag=1 ; model=$OPTARG;;  
9:*) echo ERROR $Options  
9:
```

Рис. 4.3: Выполнение командного файла №1

4.2 Задание 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. (рис. 4.4)

```
sozjyas@fedora:~ — mcedit script2.cpp
script2.cpp [----] 17 L:[ 4+18 22/ 25] *(372 / 399b) 0032 0x020 [*][X]
int main ()
{
    int n;
    cout<<"Enter your number: "<< endl;
    cin>>n;

    if (n==0){.
    cout<<"Your number is equal to 0"<<endl;
    exit(0);
    }
    if ( n>0 )
    {
    cout<<"Your number is higher than 0"<<endl;
    exit(2*n);
    }
    if (n<0)
    {
    cout<<"Your number is less than 0"<<endl;
    exit((-2*n) + 1);
    }
    return 0;
}
```

Рис. 4.4: Программа на языке Си

- Написал командный файл, который вызывает эту программу и, проанализировав с помощью команды \$?, выдает сообщение о том, какое число было введено. (рис. 4.5)

```
sozjyas@fedora:~ — mcedit script2.sh
script2.sh [----] 40 L:[ 1+14 15/ 16] *(235 / 238b) 0010 0x00A [*][X]
#!/bin/bash
g++ -o cpp script2.cpp
./cpp
a=$?
b=$a%2

if (( $a == 0 ))
then echo Your number is 0
elif (( b==0 ))
then echo Your number is "$(( $a / 2 ))"
else
let "a=$a-1"
b=-2
echo Your number is "$(( $a / $b ))"
fi
```

Рис. 4.5: Командный файл №2

- Выполнил командный файл (рис. 4.6)

```
sozjyas@fedora:~  
[sozjyas@fedora ~]$ mcedit script2.cpp  
[sozjyas@fedora ~]$ ./script2.sh  
Enter your number:  
-5  
Your number is less than 0  
Your number is -5  
[sozjyas@fedora ~]$ ./script2.sh  
Enter your number:  
4  
Your number is higher than 0  
Your number is 4  
[sozjyas@fedora ~]$ ./script2.sh  
Enter your number:  
0  
Your number is equal to 0  
Your number is 0  
[sozjyas@fedora ~]$
```

Рис. 4.6: Выполнение командного файла №2

4.3 Задание 3

- Написал 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 умеет удалять все созданные им файлы (если они существуют). (рис. 4.7)

```
sozjyas@fedora:~ — mcedit script3.sh  
script3.sh  [----]  0 L: [ 1+17 18/ 18] *(200 / 200b) <EOF>  [*][X]  
#!/bin/bash  
let a=$2  
let choice=$1  
  
if (( $choice == 0 ))  
then for (( c=1; c<=$a; c++ ))  
do  
    touch "$c".tmp  
done  
fi  
  
if (( $choice == 1 ))  
then for (( c=1; c<=$a; c++ ))  
do  
    rm "$c".tmp  
done  
fi  
█
```

Рис. 4.7: Командный файл №3

- Выполнил командный файл (рис. 4.8)

```
[sozjyas@fedora ~]$ ./script3.sh 0 5
[sozjyas@fedora ~]$ ls
1.tmp      Bureau      logfile     out2.txt    script3.sh
2.tmp      cpp         Modèles    out3.txt    scripto.cpp
3.tmp      Documents  morefun    out4.txt    script..sh
4.tmp      Images     Musique    Pictures    Téléchargements
5.tmp      _index.md  newdir     Public      Vidéos
avatar.jpg install-tl-unx newdir1    script1.sh  work
backup     'install-tl-unx (1)' ocpp      script2.cpp z
bin        install-tl-unx.tar.gz out1.txt  script2.sh

[sozjyas@fedora ~]$ ./script3.sh 1 5
[sozjyas@fedora ~]$ ls
avatar.jpg _index.md  Musique    out4.txt    scripto.cpp
backup     install-tl-unx newdir     Pictures    script..sh
bin        'install-tl-unx (1)' newdir1    Public      Téléchargements
Bureau    install-tl-unx.tar.gz ocpp      script1.sh  Vidéos
cpp        logfile    out1.txt  script2.cpp work
Documents Modèles    out2.txt  script2.sh  z
Images    morefun    out3.txt  script3.sh

[sozjyas@fedora ~]$
```

Рис. 4.8: Выполнение командного файла №2

4.4 Задание 4

- Написал 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). (рис. 4.9)

```
sozjyas@fedora:~ — mcedit script4.sh
script4.sh  [-----] 17 L:[ 1+ 4 5/ 5] *(111 / 148b) 0046 0x02E
#!/bin/bash
echo Введите полный путь директории:
dir=$1
tar -cvf new1 tar `find "$dir" -mtime 7 -type f`
```

Рис. 4.9: Командный файл №4

- Выполнил командный файл (рис. 4.10)

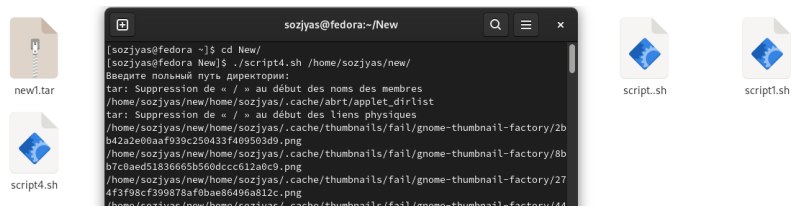


Рис. 4.10: Выполнение командного файла №2

5 Выводы

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

6 Контрольные вопросы

1. 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:LtOPTLETer do case optletter in) o flag = 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. 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.
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 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.
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 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. 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.
6. Введенная строка означает условие существования файла `man s/i.$s`
7. 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

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016.
URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. Learning the bash Shell: Unix Shell Programming. O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. Bash Pocket Reference. O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.