

Отчёт к лабораторной работе №11

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

Тимофеева Екатерина Николаевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	10
5	Выводы	13
6	Контрольные вопросы	14
	Список литературы	17

Список иллюстраций

4.1	Написание командного файла	10
4.2	Написание программы	11
4.3	Написание командного файла	11
4.4	Написание командного файла	12

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 N (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

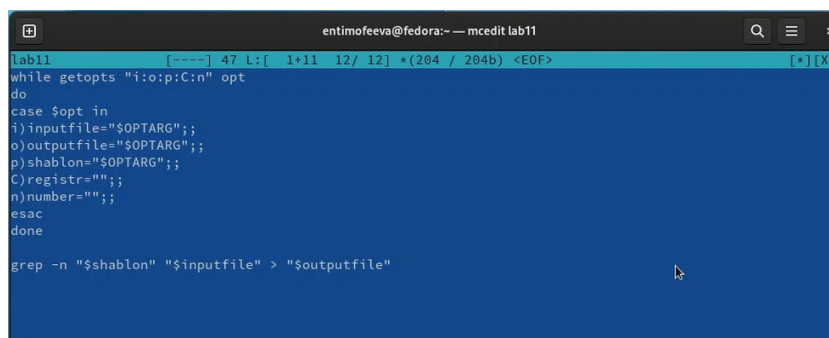
1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла;

`-ooutputfile` — вывести данные в указанный файл;

`-ршаблон` — 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`. (рис. 4.1).



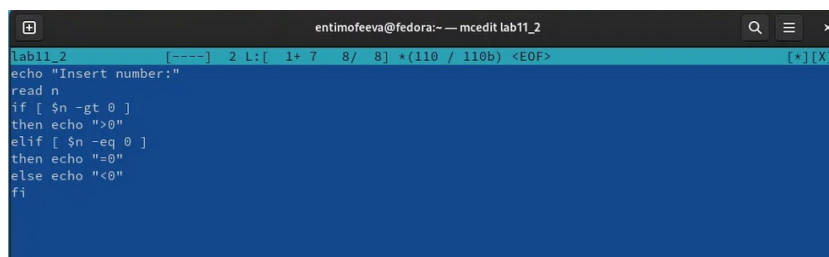
```
entimofeeva@fedora:~ — mcedit lab11
lab11 [-----] 47 L: [ 1+11 12/ 12] *(204 / 204b) <EOF> [*] [X]
while getopts "i:o:p:C:n" opt
do
case $opt in
i)inputfile="$OPTARG";;
o)outputfile="$OPTARG";;
p)shablon="$OPTARG";;
C)registr="";;
n)number="";;
esac
done

grep -n "$shablon" "$inputfile" > "$outputfile"
```

Рис. 4.1: Написание командного файла

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

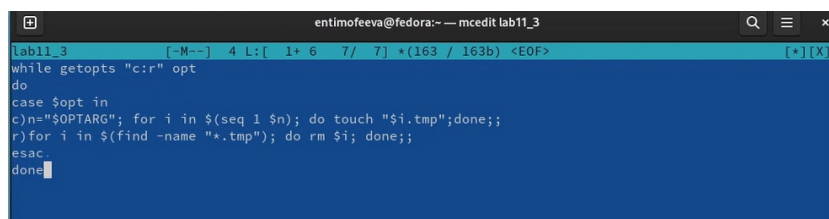
и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено. (рис. 4.2).



```
entimofeeva@fedora:~ — mcedit lab11_2
lab11_2  [----]  2 L:[ 1+ 7  8/  8] *(110 / 110b) <EOF>  [*][X]
echo "Insert number:"
read n
if [ $n -gt 0 ]
then echo ">0"
elif [ $n -eq 0 ]
then echo "=0"
else echo "<0"
fi
```

Рис. 4.2: Написание программы


3. Написать командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до N (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). (рис. 4.3).



```
entimofeeva@fedora:~ — mcedit lab11_3
lab11_3  [-M--]  4 L:[ 1+ 6  7/  7] *(163 / 163b) <EOF>  [*][X]
while getopts "c:r" opt
do
case $opt in
c)n="$OPTARG"; for i in $(seq 1 $n); do touch "$i.tmp";done;;
r)for i in $(find -name "*.tmp"); do rm $i; done;;
esac
done
```

Рис. 4.3: Написание командного файла

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`). (рис. 4.4).



The image shows a terminal window titled "entimofeeva@fedora:~ — mcedit lab11_4". The window contains a shell script named "lab11_4" with the following content:

```
lab11_4 [-M--] 0 L: [ 1+ 8 9/ 9] *(139 / 139b) <EOF> [*][X]  
while getopts "p" opt;  
do  
case $opt in  
p)dir="$OPTARG";;  
esac  
done  
find $dir -mtime +0 -mtime -7 -print0 | xargs -0 tar -cf archive.tar
```

Рис. 4.4: Написание командного файла

5 Выводы

В процессе выполнения данной лабораторной работы я изучила основы программирования в оболочке ОС UNIX. Научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

6 Контрольные вопросы

Каково предназначение команды `getopts`? 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 $optletter in
o) oflag = 1; oval =OPTARG;;
i) iflag=1; ival=$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.

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; `[c1-c2]` – соответствует любому символу, лексикографически находящемуся между символами `c1` и `c2`. Например, `echo *` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `ls .c` – выведет все файлы с последними двумя символами, совпадающими с `.c`. `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.

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.

Какие операторы используются для прерывания цикла? Два несложных спосо-

ба 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.

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 mans/i.s, ?iftest — fmans/i.s` проверяет, существует ли файл `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.

Список литературы

Лабораторная работа № 10. Программирование в командном процессоре ОС UNIX. Командные файлы (Электронный ресурс]. URL: <https://esystem.ru/dn.ru/>.