

Шаблон отчёта по лабораторной работе № 12

Операционные Системы

Адебайо Ридвануллахи Айофе

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	16
5	Контрольные вопросы	17

List of Tables

List of Figures

3.1	mcredit	8
3.2	mcredit	9
3.3	ВЫВОД	10
3.4	mcredit	11
3.5	mcredit	11
3.6	ВЫВОД	12
3.7	mcredit	13
3.8	ВЫВОД	14
3.9	mcredit	15
3.10	ВЫВОД	15

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 Выполнение лабораторной работы

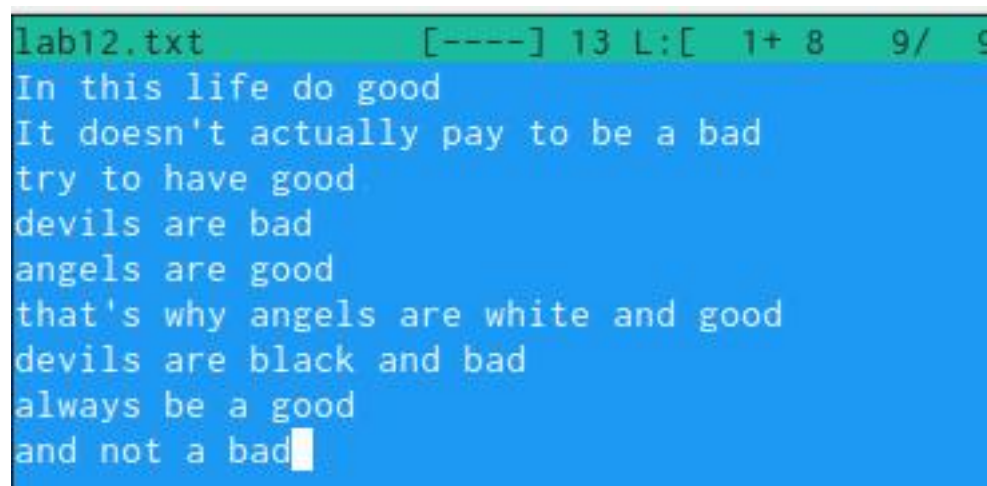
1. Используя команды `getopts` `grep`, написать командный файл, который анализирует командную строку с ключами: `-iinputfile` — прочитать данные из указанного файла; `-ooutputfile` — вывести данные в указанный файл; `-rшаблон` — указать шаблон для поиска; `-C` — различать большие и малые буквы; `-n` — выдавать номера строк. а затем ищет в указанном файле нужные строки, определяемые ключом `-p`.

The image shows a screenshot of the mcedit editor window. The title bar reads "AA: mcedit — Konsole". The menu bar includes "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The status bar at the top shows "lab12.sh", "[-M--]", "27 L:", "1+30", "31/ 35", "*(696 / 767b)", "0010 0x00A", and "[*][X]". The main text area contains a shell script with the following content:

```
#!/bin/bash
while getopts i:o:p:Cn optletter
do case $optletter in
<----->i) iflag=1 ival=$OPTARG;;
<----->o) oflag=1 oval=$OPTARG;;
<----->p) pflag=1 pval=$OPTARG;;
<----->C) Cflag=1;;
<----->n) nflag=1;;
<----->*) echo Illegal option $optletter
    esac
done
if (((Cflag==1)&&(nflag==1)))
then grep -e${pval} -i -n ${ival}
    if ((oflag==1))
    then grep -e${pval} -i -n ${ival} >${oval}
    fi
fi
if (((Cflag==1)&&(nflag==0)))
then grep -e${pval} -i ${ival}
    if ((oflag==1))
    then grep -e${pval} -i ${ival} > ${oval}
    fi
fi
if (((Cflag==0)&&(nflag==1)))
then grep -e${pval} -n ${ival}
    if((oflag==1))
    then grep -e${pval} -n ${ival} > ${oval}
    fi
fi
if (((Cflag==0)&&(nflag==0)))
then grep -e${pval} ${ival}
    if((oflag==1))
    then grep -e${pval} ${ival} > ${oval}
    fi
fi
```

The bottom status bar contains a menu with 10 items: 1Помощь, 2Сохранить, 3Блок, 4Замена, 5Копия, 6Переместить, 7Поиск, 8Удалить, 9МенюМС, 10Выход.

Figure 3.1: mcedit



```
lab12.txt      [----] 13 L:[ 1+ 8  9/ 9
In this life do good
It doesn't actually pay to be a bad
try to have good
devils are bad
angels are good
that's why angels are white and good
devils are black and bad
always be a good
and not a bad
```

Figure 3.2: mcedit

```
AA: bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
raadebayjo@dk6n64 ~/AA $ mcedit lab12.txt
raadebayjo@dk6n64 ~/AA $ mcedit lab12.sh
raadebayjo@dk6n64 ~/AA $ mcedit lab12.txt
raadebayjo@dk6n64 ~/AA $ ./lab12.sh -ilab12.txt -olab12_1.txt -pood
bash: ./lab12.sh: Отказано в доступе
raadebayjo@dk6n64 ~/AA $ chmod +x lab12.sh
raadebayjo@dk6n64 ~/AA $ ./lab12.sh -ilab12.txt -olab12_1.txt -pood
In this life do good
try to have good
angels are good
that's why angels are white and good
always be a good
raadebayjo@dk6n64 ~/AA $ ./lab12.sh -ilab12.txt -olab12_1.txt -pood -n
1:In this life do good
3:try to have good
5:angels are good
6:that's why angels are white and good
8:always be a good
raadebayjo@dk6n64 ~/AA $ ./lab12.sh -ilab12.txt -olab12_1.txt -pood -C -n
1:In this life do good
3:try to have good
5:angels are good
6:that's why angels are white and good
8:always be a good
raadebayjo@dk6n64 ~/AA $ ./lab12.sh -ilab12.txt -olab12_1.txt -pood -C
In this life do good
try to have good
angels are good
that's why angels are white and good
always be a good
raadebayjo@dk6n64 ~/AA $
```

Figure 3.3: вывод

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 код завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено.

```

lab12.c      [-M--]  1
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a;
    printf("Input a: ");
    scanf("%i", &a);
    if (a==0) exit (0);
    else if (a<0) exit (1);
    else if (a>0) exit (2);
    return (3);
}

```

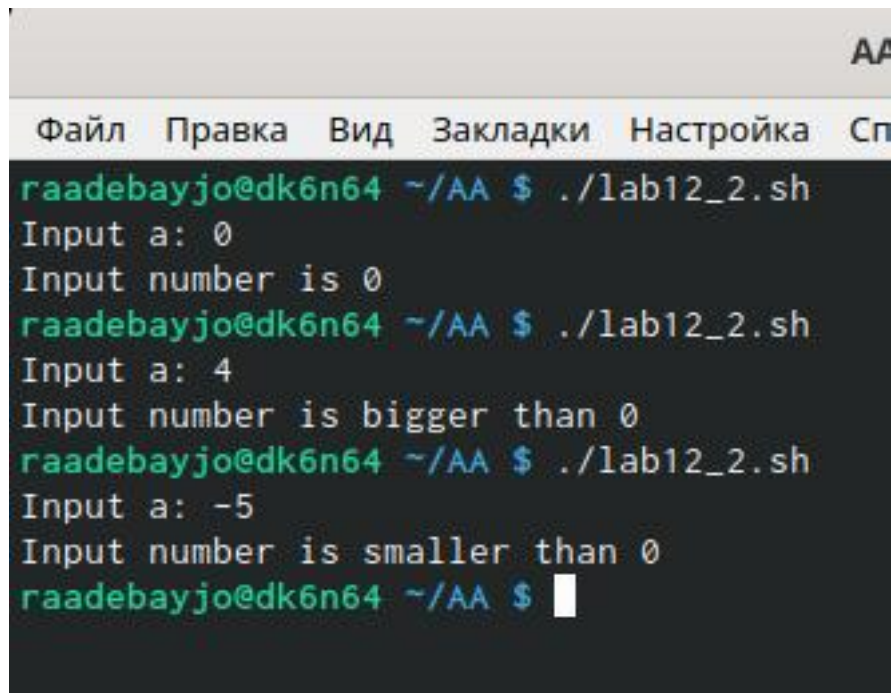
Figure 3.4: mcedit

```

lab12_2.sh    [----]  4 L:[ 1+ 7  8/  8]
#!/bin/bash
gcc -c cprog lab12.c
./cprog
case $? in
    0) echo "Input number is 0";;
    1) echo "Input number is smaller than 0";;
    2) echo "Input number is bigger than 0";;
esac

```

Figure 3.5: mcedit



```
raadebayjo@dk6n64 ~/AA $ ./lab12_2.sh
Input a: 0
Input number is 0
raadebayjo@dk6n64 ~/AA $ ./lab12_2.sh
Input a: 4
Input number is bigger than 0
raadebayjo@dk6n64 ~/AA $ ./lab12_2.sh
Input a: -5
Input number is smaller than 0
raadebayjo@dk6n64 ~/AA $
```

Figure 3.6: вывод

3. Написал 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).

```

lab12_3.sh      [-M--]  2 L:[
#!/bin/bash
let dflag=0
while getopts a:d optletter
do case $optletter in
a) aflag=1; aval=$OPTARG;;
d) dflag=1;;
*) echo Incorrect input $optletter
esac
done
if ((dflag==0))
then for ((i=1; i<=aval; i++))
do echo ${i}.tmp
done
fi
if ((dflag==1))
then for ((i=1; i<=aval; i++))
do rm ${i}.tmp
done
fi

```

Figure 3.7: mcedit

```
AA : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
raadebayjo@dk6n64 ~/AA $ touch lab12_3.sh
raadebayjo@dk6n64 ~/AA $ chmod +x lab12_3.sh
raadebayjo@dk6n64 ~/AA $ mcedit lab12_3.sh

raadebayjo@dk6n64 ~/AA $ ./lab12_3.sh -a10
raadebayjo@dk6n64 ~/AA $ ls
10.tmp  3.tmp  6.tmp  9.tmp  feathers  lab12_3.sh  lab12.txt
1.tmp   4.tmp  7.tmp  australia  lab12_1.txt  lab12.c  my_os
2.tmp   5.tmp  8.tmp  cprog     lab12_2.sh  lab12.sh  play
raadebayjo@dk6n64 ~/AA $ ./lab12_3.sh -a10 -d
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
gm: невозможно удалить '.tmp': Нет такого файла или каталога
raadebayjo@dk6n64 ~/AA $ ls
10.tmp  3.tmp  6.tmp  9.tmp  feathers  lab12_3.sh  lab12.txt
1.tmp   4.tmp  7.tmp  australia  lab12_1.txt  lab12.c  my_os
2.tmp   5.tmp  8.tmp  cprog     lab12_2.sh  lab12.sh  play
raadebayjo@dk6n64 ~/AA $ mcedit lab12_3.sh

raadebayjo@dk6n64 ~/AA $ ./lab12_3.sh -a10 -d
raadebayjo@dk6n64 ~/AA $ ls
australia  feathers  lab12_2.sh  lab12.c  lab12.txt  play
cprog     lab12_1.txt  lab12_3.sh  lab12.sh  my_os
raadebayjo@dk6n64 ~/AA $
```

Figure 3.8: вывод

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).


```
lab12_4.sh      [-M--] 54 L:[ 1+ 2  3/ 3] *(89 /
#!/bin/bash
tar -cf lab12_4.tar $@
find $@ -mtime -7 -exec tar -rf lab12_4modif.tar {} \;
```

Figure 3.9: mcedit

```
raadebayjo@dk6n64 ~/AA $ ./lab12_4.sh os-intro
tar: os-intro: Функция stat завершилась с ошибкой: Нет такого файла или каталога
tar: Завершение работы с состоянием неисправности из-за возникших ошибок
find: 'os-intro': Нет такого файла или каталога
raadebayjo@dk6n64 ~/AA $ ls
australia  feathers      lab12_2.sh  lab12_4.sh  lab12.c    lab12.txt  play
cprog      lab12_1.txt  lab12_3.sh  lab12_4.tar  lab12.sh   my_os
raadebayjo@dk6n64 ~/AA $
```

Figure 3.10: вывод

4 Выводы

В результате работы, научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 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 -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) 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. Единственная функция этой
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`