Цель работы

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

Задание

Написать командные файлы, выполняющие указанные действия

Ход работы

Необходимо выполнить следующие задания

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;
   – **-pшаблон** — 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.

   - Командный файл анализирует введенные ключи командой **getopts**, присваивая имя входного и выходного файлов и шаблон для поиска соответствующим переменным и определяет, нужно ли использовать ключи **-i** и **-n** команды **grep**. Если введены некорректные ключи, то выводится ошибка. Ошибка также выводится, если не задан входной и выходной файлы или шаблон для поиска. В конце команда **grep** выводит из входного файла в выходной файл результаты поиска по заданному шаблону и с заданными параметрами (Рис.1)

```
task1.sh           [-M--] 115 L:[  1+25  26/ 26] *(1455/1455b) <EOF>
nflag="" <------>#Присвоение переменной nflag пустой строки
Cflag="-i"<----->#Присвоение переменной Cflag оператора -i команды grep
while getopts i:o:p:Cn optletter; do<-->#Чтение всех операторов
    case $optletter in
<------>i) ival=$OPTARG;;<----->#Присвоение переменной имени входного файла
<------>o) oval=$OPTARG;;<----->#Присвоение переменной имени выходного файла
<------>p) pval=$OPTARG;;<----->#Присвоение переменной шаблона поиска
<------>C) Cflag="";;<->#Удаление из переменной оператора -i
<------>n) nflag="-n";;<------>#Присвоение оператора -n команды grep для вывода номера строки
<------>*) echo "Illegal option $optletter"<--->#Вывод ошибки при попытке ввести непредусмотренные операторы
    esac
done

if [ -z $ival ]; then
    echo "No input file"; exit<>#Следующие три if - сообщения об ошибках, связанных с недостаточностью введенных данных
fi

if [ -z $oval ]; then
    echo "No output file"; exit
fi

if [ -z $pval ]; then
    echo "No pattern"; exit
fi

grep $nflag $Cflag $pval $ival > $oval<>#Вывод в файл output результата работы команды grep с заданными параметрами
```

Рис.1

- Результат выполнения программы (предварительно командой **chmod +x task1.sh** было дано разрешение на выполнение)(Рис.2)

```
[dnbabkov@dnbabkov lab12]$ ./task1.sh -iinput -ooutput -ptest -C -n
[dnbabkov@dnbabkov lab12]$ cat output
2:test
```

Рис.2

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

- Сначала напишем программу на языке С, выполняющую то, что требуется в задании. (Рис.3)

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

int main(){

    int n;

    scanf("%d", &n);

    if (n < 0) {
<------>exit(1);
    }

    if (n > 0) {
<------>exit(2);
    }

    exit(0);
}
```

Рис.3

- Создаем командный файл, который принимает код завершения программы, написанной ранее, и в соответствии с кодом завершения выводит сообщение о введенном числе (Рис.4)

```
task2.sh            [----] 70 L:[  1+ 7   8/  8] *(388 / 388b) <EOF>
./prog<><------>#Запуск программы на языке С
case $? in<---->#Анализ кода завершения программы
0)<---->s="равно нулю.";;<----->#Соответствующие выводы
1)<---->s="меньше нуля.";;
2)<---->s="больше нуля.";;
esac

echo "Ведённое число $s"<------>#Вывод с соответствующим числу выводом
```

Рис.4

- Результат выполнения программы (предварительно командой **chmod +x task2.sh** было дано разрешение на выполнение) (Рис.5)

```
[dnbabkov@dnbabkov lab12]$ ./task2.sh
0
Ведённое число равно нулю.
[dnbabkov@dnbabkov lab12]$ ./task2.sh
12
Ведённое число больше нуля.
[dnbabkov@dnbabkov lab12]$
[dnbabkov@dnbabkov lab12]$ ./task2.sh
-12
Ведённое число меньше нуля.
[dnbabkov@dnbabkov lab12]$
```

Рис.5

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).

   - Командный файл принимает на вход ключ **-счисло** или ключ **-d** и в соответствии с этим создает указанное число файлов или удаляет их (Рис.6)

```
task3.sh           [----] 80 L:[  1+17  18/ 20] *(619 / 636b) 0010 0x00A
num=0
while getopts c:d ol<-->#Поиск операторов
do
case $ol in
    c)num=$OPTARG;;<--->#Кол-во файлов, которое нужно создать
    d)del=true;;<------>#Файлы должны быть удалены
    *)echo "Wait, that's illegal!"<---->#Анализ некорректных операторов
esac
done

if [ $num -gt 0 ]<----->#Создание файлов, если кол-во не равно нулю
then for (( i=1; i<=$num; i++ ))
    do
<------>touch ${i}.tmp
    done
fi

if [ "$del" = true ]<-->#Удаление всех файлов с расширением .tmp, если это нужно
then rm *.tmp
fi
```

Рис.6

   - Результат выполнения программы (предварительно командой **chmod +x task3.sh** было дано разрешение на выполнение) (Рис.7)

```
[dnbabkov@dnbabkov lab12]$ ./task3.sh -c7
[dnbabkov@dnbabkov lab12]$ ls -l
total 64
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 26 21:10 1
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 1.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 2.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 3.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 4.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 5.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 6.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 29 00:06 7.tmp
-rw-rw-r--. 1 dnbabkov dnbabkov 3889 May 27 21:48 dirArch.tar.gz
-rw-rw-r--. 1 dnbabkov dnbabkov   31 May 26 21:08 input
-rw-rw-r--. 1 dnbabkov dnbabkov    7 May 28 23:58 output
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:34 prog
-rw-rw-r--. 1 dnbabkov dnbabkov  169 May 27 20:34 prog.c
-rw-rw-r--. 1 dnbabkov dnbabkov  171 May 27 20:31 prog.cpp
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:25 prog.o
-rwxrwxr-x. 1 dnbabkov dnbabkov 1455 May 28 23:57 task1.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  388 May 29 00:00 task2.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  636 May 29 00:05 task3.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  202 May 27 21:48 task4.sh
-rwxr-xr-x. 1 dnbabkov dnbabkov  508 May 26 20:23 test.sh
[dnbabkov@dnbabkov lab12]$ ./task3.sh -d
[dnbabkov@dnbabkov lab12]$ ls -l
total 64
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 26 21:10 1
-rw-rw-r--. 1 dnbabkov dnbabkov 3889 May 27 21:48 dirArch.tar.gz
-rw-rw-r--. 1 dnbabkov dnbabkov   31 May 26 21:08 input
-rw-rw-r--. 1 dnbabkov dnbabkov    7 May 28 23:58 output
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:34 prog
-rw-rw-r--. 1 dnbabkov dnbabkov  169 May 27 20:34 prog.c
-rw-rw-r--. 1 dnbabkov dnbabkov  171 May 27 20:31 prog.cpp
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:25 prog.o
-rwxrwxr-x. 1 dnbabkov dnbabkov 1455 May 28 23:57 task1.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  388 May 29 00:00 task2.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  636 May 29 00:05 task3.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  202 May 27 21:48 task4.sh
-rwxr-xr-x. 1 dnbabkov dnbabkov  508 May 26 20:23 test.sh
```

Рис.7

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).
    - Командный файл принимает на вход имя директории. Если имени нет, по выводится сообщение о том, что имени нет, и выполнение прекращается. Командой **tar** с ключом **-**

zcvf все файлы из заданной директории архивируются в файл **dirArch.gz** (Рис.8)

```
task4.sh            [----] 28 L
path=$1

if [ -z $path ]; then
    echo "No path"
    exit
fi

tar -zcvf dirArch.gz $path/*
```

Рис.8

- Результат выполнения программы (предварительно командой **chmod +x task4.sh** было дано разрешение на выполнение) (Рис.9)

```
[dnbabkov@dnbabkov lab12]$ ./task4.sh .
./1
./input
./output
./prog
./prog.c
./prog.cpp
./prog.o
./task1.sh
./task2.sh
./task3.sh
./task4.sh
./test.sh
[dnbabkov@dnbabkov lab12]$ ls -l
total 64
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 26 21:10 1
-rw-rw-r--. 1 dnbabkov dnbabkov 3799 May 27 21:13 dirArch.gz
-rw-rw-r--. 1 dnbabkov dnbabkov   31 May 26 21:08 input
-rw-rw-r--. 1 dnbabkov dnbabkov   14 May 26 21:29 output
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:34 prog
-rw-rw-r--. 1 dnbabkov dnbabkov  169 May 27 20:34 prog.c
-rw-rw-r--. 1 dnbabkov dnbabkov  171 May 27 20:31 prog.cpp
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:25 prog.o
-rwxrwxr-x. 1 dnbabkov dnbabkov  426 May 26 21:28 task1.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  155 May 27 20:40 task2.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  249 May 27 21:09 task3.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov   91 May 27 21:12 task4.sh
-rwxr-xr-x. 1 dnbabkov dnbabkov  508 May 26 20:23 test.sh
```

Рис.9

- Модифицируем командный файл, чтобы он архивировал только те файлы, которые были изменены менее недели назад (плюс чтобы он не пытался заархивировать сам себя). Для этого используем команду **find** с ключом **-mtime -7** и **-not -name "$arcName"** (Рис.10)

```
task4.sh            [----] 166 L:[  1+ 9  10/ 12] *(545 / 585b) 0010 0x00A
path=$1<->#Имя директории, файлы которой нужно запаковать
arcName="dirArch.tar"

if [ -z $path ]; then<->#Проверка на наличие пути
    echo "No path"
    exit
fi

tar -cvf $arcName -T /dev/null<>#Создание пустого архива
tar -rvf $arcName `find $path -type f -mtime -7 -not -name "$arcName*"` >#Архивирование файлов директории, которые были изменены менее недели назад, исключая сам архив

gzip $arcName<->#Сжатие архива
```

Рис.10

- Результат выполнения программы (предварительно командой **chmod +x task4.sh** было дано разрешение на выполнение) (Рис.11)

```
[dnbabkov@dnbabkov lab12]$ ./task4.sh .
./input
./output
./test.sh
./task1.sh
./1
./prog.c
./prog.o
./prog.cpp
./prog
./task2.sh
./task3.sh
./task4.sh
gzip: dirArch.tar.gz already exists; do you wish to overwrite (y or n)? y
[dnbabkov@dnbabkov lab12]$ ls -l
total 64
-rw-rw-r--. 1 dnbabkov dnbabkov    0 May 26 21:10 1
-rw-rw-r--. 1 dnbabkov dnbabkov 3889 May 27 21:48 dirArch.tar.gz
-rw-rw-r--. 1 dnbabkov dnbabkov   31 May 26 21:08 input
-rw-rw-r--. 1 dnbabkov dnbabkov   14 May 26 21:29 output
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:34 prog
-rw-rw-r--. 1 dnbabkov dnbabkov  169 May 27 20:34 prog.c
-rw-rw-r--. 1 dnbabkov dnbabkov  171 May 27 20:31 prog.cpp
-rwxrwxr-x. 1 dnbabkov dnbabkov 8416 May 27 20:25 prog.o
-rwxrwxr-x. 1 dnbabkov dnbabkov  426 May 26 21:28 task1.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  155 May 27 20:40 task2.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  249 May 27 21:09 task3.sh
-rwxrwxr-x. 1 dnbabkov dnbabkov  202 May 27 21:48 task4.sh
-rwxr-xr-x. 1 dnbabkov dnbabkov  508 May 26 20:23 test.sh
```

Рис.11

Контрольные вопросы

1. Команда 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, для команды ls флагом может являться -F. 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.
   Функция getopts включает две специальные переменные среды – OPTARG и OPTIND. Если ожидается дополнительное значение, то OPTARG устанавливается в значение этого аргумента. Функция getopts также понимает переменные типа массив, следовательно, можно использовать её в функции не только для синтаксического анализа аргументов функций, но и для анализа введённых пользователем данных.
2. 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.

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 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. 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. 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).
   Примеры бесконечных циклов:

```
while true
    do echo hello andy
done
until false
    do echo hello mike
done
```

6. Строка **if test -f man$s/$i.$s** проверяет, существует ли файл **man$s/$i.$s** и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернет нулевое значение (ложь).
7. 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.

Вывод

В ходе выполнения лабораторной работы я научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.