



26.5.2022

Лабораторная работа 11

Манатов Рамазан Русланович

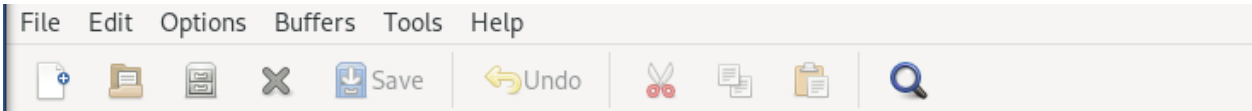
Цель работы: Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

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;
- `-р`шаблон — 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` (image/1.png)

Для данной задачи создадим файл `prog1.sh` и напомним соответствующие скрипты (рис1)



```
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
while getopts i:o:p:C:n 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 $optletter
    esac
done
if (($pflag==0))
then echo "Шаблон не найден"
else
    if (($iflag==0))
    then echo "don't found file"
    else
        if (($oflag==0))
        then if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival
                else grep -n $pval $ival
                fi
            else if (($nflag==0))
                then grep -i $pval $ival
                else grep -i -n $pval $ival
                fi
            fi
        else if (($cflag==0))
            then if (($nflag==0))
                then grep $pval $ival > $oval
                else grep -n $pval $ival > $oval
                fi
            else if (($nflag==0))
                then grep -i $pval $ival > $oval
                else grep -i -n $pval $ival > $oval
                fi
            fi
        fi
    fi
fi
```

Рисунок 1 код программы

Далее я проверил работу написанного скрипта ,используя различные опции предварительно добавив право на исполнение файла командой `chmod +x` и создав 2 файла ,которые необходимы для выполнения программ (рис2)

```
[rrManatov@fedora ~]$ chmod +x progl.sh
[rrManatov@fedora ~]$ cat a1.txt
Moscow is the capital of Russia
Paris is the Capital of France
Simple text
Rome is not the CAPITAL of Canada
[rrManatov@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p capital -C -n
./progl.sh: строка 3: getopts: команда не найдена
Шаблон не найден
[rrManatov@fedora ~]$ emacs
[rrManatov@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p capital -C -n
illegal option i
illegal option o
```

```
[rrManatov@fedora ~]$ cat a2.txt
Moscow is the capital of Russia
Paris is the Capital of France
Simple text
Rome is not the CAPITAL of Canada
[rrManatov@fedora ~]$ ./progl.sh -i a1.txt -o a2.txt -p capital -C -n
illegal option i
illegal option o
illegal option p
illegal option C
illegal option n
```

```
[rrManatov@fedora ~]$ cat a2.txt
1:Moscow is the capital of Russia
2:Paris is the Capital of France
4:Rome is not the CAPITAL of Canada
[rrManatov@fedora ~]$ ./progl.sh -i a1.txt -C -n
illegal option i
illegal option C
illegal option n
Шаблон не найден
[rrManatov@fedora ~]$ ./progl.sh -o a2.txt -p capital -C -n
illegal option o
illegal option p
illegal option C
illegal option n
Шаблон не найден
```

Рисунок 2 проверка работы кода

Скрипт работает корректно

2) Напишем 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, проанализировать с помощью команды `$?`

Для данной задачи я создал два файла `chislo.c` и `chislo.sh` и написал нужный код (рис3) (рис4) (рис5) (рис6)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Введите число/n");
    int a;
    scanf("%d", &a);
    if (a < 0) exit(0);
    if (a > 0) exit(1);
    if (a == 0) exit(2);
    return 0;
}
```

Рисунок 3 код программы 1

```
#!/bin/bash
gcc chislo.c -o chislo
./chislo
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число больше 0";;
    2) echo "Число равно 0"
esac
```

Рисунок 4 коде программы 2

Далее проверим работу программы предварительно добавив право на выполнение

```
[rrManatov@fedora ~]$ chmod +x chislo.sh
```

Рисунок 5 добавили право на выполнение

```
[rrManatov@fedora ~]$ ./chislo.sh
Введите число/n
1
Число больше 0
[rrManatov@fedora ~]$ ./chislo.sh
Введите число/n
-1
Число меньше 0
[rrManatov@fedora ~]$ ./chislo.sh
Введите число/n
0
Число равно 0
```

Рисунок 6 программа работает корректно

3) Напишем 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). (рис7)

Для данной задачи я создал файл files.sh и написал соответствующий код

```
#!/bin/bash
opt=$1;
format=$2;
number=$3;
function Files()
{
    for ((i=1; i<=$number; i++)) do
        file=$(echo $format | tr '#' "$i")
        if [ $opt == "-r" ]
        then
            rm -f $file
        elif [ $opt == "-c" ]
        then
            touch $file
        fi
    done
}
Files
```

Рисунок 7 код программы

Далее проверим работу написанного файла предварительно добавив право на исполнение

Сначала я создал три файла удовлетворяющих условию задачи и потом удалил их

```
[rrManatov@fedora ~]$ emacs
[rrManatov@fedora ~]$ chmod +x files.sh
[rrManatov@fedora ~]$ ls
a1.txt      chislo.sh  file.txt   os-intro   work_
a1.txt~     chislo.sh~ file12.doc prog1.sh    Видео
a2.txt      code2.sh  file1.doc  prog1.sh~  Документы
a2.txt~     code2.sh~ format.sh  progl.sh   Загрузки
abc1        conf.txt  format.sh~ progl.sh~  Изображения
backup      '~exit'   '#lab.04PRO#' ski.plases Музыка
backup.sh   file.cpp  lab.04PRO  '#text.txt#' Общедоступные
backup.sh~  file.old. lab.04.PRO text.txt   'Рабочий стол'
chislo      file.pdf  lab05      tmp        Шаблоны
chislo.c    files.sh  lab07      work
chislo.c~   files.sh~ lab09      work.
```

```
[rrManatov@fedora ~]$ ./files.sh -c abc#.txt 3
[rrManatov@fedora ~]$ ls
a1.txt      chislo.c    files.sh~  os-intro   Видео
a1.txt~     chislo.c~   file.txt   prog1.sh    Документы
a2.txt      chislo.sh  file12.doc prog1.sh~  Загрузки
a2.txt~     chislo.sh~ file1.doc  progl.sh   Изображения
abc1        code2.sh  format.sh  progl.sh~  Музыка
abc1.txt    code2.sh~ format.sh~ ski.plases Общедоступные
abc2.txt    conf.txt   '#lab.04PRO#' '#text.txt#' 'Рабочий стол'
abc3.txt    '~exit'   lab.04PRO  text.txt   Шаблоны
backup      file.cpp   lab.04.PRO tmp
backup.sh   file.old. lab05      work
backup.sh~  file.pdf  lab07      work.
chislo      files.sh  lab09      work_
```

```
[rrManatov@fedora ~]$ ./files.sh -r abc#.txt 3
[rrManatov@fedora ~]$ ls
a1.txt      chislo.sh  file.txt   os-intro   work_
a1.txt~     chislo.sh~ file12.doc prog1.sh    Видео
a2.txt      code2.sh  file1.doc  prog1.sh~  Документы
a2.txt~     code2.sh~ format.sh  progl.sh   Загрузки
abc1        conf.txt  format.sh~ progl.sh~  Изображения
backup      '~exit'   '#lab.04PRO#' ski.plases Музыка
backup.sh   file.cpp  lab.04PRO  '#text.txt#' Общедоступные
backup.sh~  file.old. lab.04.PRO text.txt   'Рабочий стол'
chislo      file.pdf  lab05      tmp        Шаблоны
chislo.c    files.sh  lab07      work
chislo.c~   files.sh~ lab09      work.
```

```
[rrManatov@fedora ~]$
```

Рисунок 8 Программа работает корректно

4) Напишем командный файл, который с помощью команды `tar` запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду `find`)

Для данной задачи я создал файл `prog4.sh` предварительно добавив право на выполнение `chmod +x`

```
#!/bin/bash
files=$(find ./ -maxdepth 1 -mtime -7)
Listing=""
for file in "$files" ; do
    file=$(echo "$file" | cut -c 3-)
    listing="$listing $file"
done
dir=$(basename $(pwd))
tar -cvf $dir.tar $listing
```

Рисунок 9 код программы

Теперь проверим корректность скрипта

```
[rrManatov@fedora Catalog1]$ ls -l
итого 0
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a1
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a2
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a3
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a4
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a5
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a6
-rw-rw-r--. 1 rrManatov rrManatov 0 мая 28 14:58 a7
[rrManatov@fedora Catalog1]$ sudo ~/prog4.sh
[sudo] пароль для rrManatov:
a1
a2
a3
a4
a5
a6
a7
[rrManatov@fedora Catalog1]$ tar -tf Catalog1.tar
a1
a2
a3
a4
a5
a6
a7
```

Рисунок 10 код работает корректно

Вывод: В ходе выполнения данной лабораторной работы я изучил основы программирования в оболочке ОС linux и научился писать более сложные командные файлы с использованием логических управляющих конструкций циклов

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