

## 11 Лабораторная работа

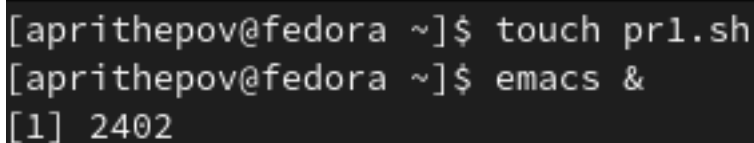
Прищепов Александр

### Цель работы

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

### Выполнение лабораторной работы

1. Создадим командный файл pr1.sh, далее откроем его в emacs (рис.1)  
рис 1:



```
[aprithepov@fedora ~]$ touch pr1.sh  
[aprithepov@fedora ~]$ emacs &  
[1] 2402
```

изображение

- Используя команды getopts grep, запишем в файл программу (рис.2), которая анализирует командную строку с ключами: - -inputfile — прочитать данные из указанного файла; - -ooutputfile — вывести данные в указанный файл; - -ршаблон — указать шаблон для поиска; - -С — различать большие и малые буквы; - -n — выдавать номера строк; а затем ищет в указанном файле нужные строки, определяемые ключом -р.

рис 2:

```
#!/bin/bash
iflag=0; oflag=0; pflag=0; cflag=0; nflag=0;
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 $optletter
    esac
done
if (($pflag==0))
then echo "Шаблон не найден"
else
    if (($iflag==0))
    then echo "Файл не найден"
    else
        if (($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
fi
```

*изображение*

- Проверим работу файла, предварительно дав ему права на выполнение (chmod +x \*.sh), и создадим два файла для проверки работы (touch one.txt two.txt) (рис.3,4). Запускаем файл

рис 3:

```
[aprithepov@fedora ~]$ chmod +x *.sh
[1]+  Завершён      emacs
[aprithepov@fedora ~]$ touch one.txt
[aprithepov@fedora ~]$ touch two.txt
[aprithepov@fedora ~]$ mcedit one.txt

[aprithepov@fedora ~]$ mcedit two.txt
```

*изображение*

рис 4:

```
[aprithepov@fedora ~]$ ./pr1.sh -i one.txt -o two.txt -p second -C -n
[aprithepov@fedora ~]$ cat two.txt
2:program for second file
4:checking second file
```

*изображение*

2. Создадим два файла для третьего задания (команда `touch pr2.c pr2.sh`) и откроем в `emacs`. Затем 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 о коде завершения в оболочку. Командный файл должен вызывать эту программу и, проанализировав с помощью команды `$?`, выдать сообщение о том, какое число было введено

```
#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;
}
```

*изображение*

```
#!/bin/bash
gcc pr2.c -o pr2
./pr2
code=$?
case $code in
    0) echo "Число меньше 0";;
    1) echo "Число равно 0";;
    2) echo "Число больше 0"
esac
```

*изображение*

- Проверим работу командного файла, передав ему права на выполнения и запустив его

```

[aprithepov@fedora ~]$ chmod +x pr2.sh
[2]+  Завершён      emacs
[aprithepov@fedora ~]$ ./pr2.sh
Введите число
6
Число больше 0
[aprithepov@fedora ~]$ touch files.sh
[aprithepov@fedora ~]$ emacs &
[1] 3943
[aprithepov@fedora ~]$ chmod +x files.sh
[1]+  Завершён      emacs
[aprithepov@fedora ~]$ ./files.sh -C #.txt 4
./files.sh: строка 7: ((: i<=: синтаксическая ошибка: ожидается операнд (невер-
ный маркер «<=»)
[aprithepov@fedora ~]$ ./files.sh -c abc#.txt 4
[aprithepov@fedora ~]$ ls

```

изображение

```

[aprithepov@fedora ~]$ ls
1      bin      pr1.sh  pr2.sh  Видео      Общедоступные
abc1.txt files.sh pr1.sh~ pr2.sh~  Документы  'Рабочий стол'
abc2.txt files.sh~ pr2     ski.places. Загрузки   Шаблоны
abc3.txt Lab-work pr2.c   two.txt  Изображения
abc4.txt one.txt  pr2.c~  work     Музыка

[aprithepov@fedora ~]$ touch pr4.sh
[aprithepov@fedora ~]$ emacs &
[1] 4036
[aprithepov@fedora ~]$ chmod +x pr4.sh
[1]+  Завершён      emacs
[aprithepov@fedora ~]$ cd catalog
bash: cd: catalog: Нет такого файла или каталога
[aprithepov@fedora ~]$ mkdir catalog
[aprithepov@fedora ~]$ cd catalog

```

изображение

3. Создадим командный файл files.sh и откроем его в emacs. Напишем командный файл, создающий указанное число файлов, пронумерованных последовательно от 1 до  $N$  (например 1.tmp, 2.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).

```
#!/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
```

*изображение*

- Проверим его работу, передав ему права на выполнения и запустив его (команда ./files.sh)
- 4. Создадим командный файл pr4.sh и откроем его в emacs. 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).

```
#!/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
```

*изображение*

- Создадим в домашнем каталоге каталог catalog и перенесем туда некоторые файлы, измененные в разное время. Дадим командному файлу право на выполнение (chmod +x pr4.sh) и запустим его в этом каталоге. Файл работает исправно.

```
[aprithepov@fedora ~]$ mkdir catalog
[aprithepov@fedora ~]$ cd catalog
[aprithepov@fedora catalog]$ ls
abc1.txt  abc3.txt  files.sh  pr1.sh  pr2.c  pr4.sh
abc2.txt  abc4.txt  one.txt  pr2      pr2.sh  two.txt
[aprithepov@fedora catalog]$ ~/pr4.sh
bash: /home/aprithepov/pr4.sh: Нет такого файла или каталога
[aprithepov@fedora catalog]$ cd
[aprithepov@fedora ~]$ cd catalog
[aprithepov@fedora catalog]$ ~/pr4.sh
one.txt
Presentation7.md
```

*изображение*

## Выводы

Я научился писать более сложные командные файлы с использованием логических управляющих конструкций и 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 введенные 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 введенных 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, `1.1 echo -` выведет имена всех файлов текущего каталога, что представляет собой простейший аналог команды `ls`; `1.2. ls.c-` выведет все файлы с последними двумя символами, совпадающими с `c.` `1.3. echo prog.?` – выведет все файлы, состоящие из пяти или шести символов, первыми пятью символами которых являются `prog.` `1.4. [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, неравный 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 mani.s` и является ли этот файл обычным файлом. Если данный файл является каталогом, то команда вернёт 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.