

Отчет по лабораторной работе №11

Операционные системы

Газизова Регина

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	12
5	Контрольные вопросы	13

Список иллюстраций

3.1	Создание файла	7
3.2	Командный файл	8
3.3	Права доступа	8
3.4	Выполнение	9
3.5	Программы	9
3.6	Выполнение	10
3.7	Командный файл	10
3.8	Выполнение	10
3.9	Командный файл	11
3.10	Выполнение	11

Список таблиц

1 Цель работы

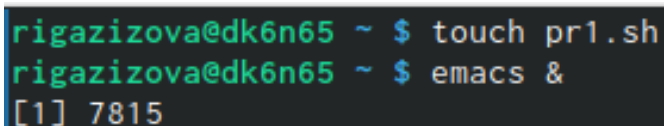
Изучить основы программирования в оболочке ОС UNIX. Научится писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

2 Задание

1. Используя команды `grep`, написать командный файл, который анализирует командную строку с ключами: `--inputfile` — прочитать данные из указанного файла; `--outputfile` — 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 `-r`.
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 `tar` запаковывает 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. Создадим командный файл pr1.sh (рис.1) и откроем его в emacs.



```
rigazizova@dk6n65 ~ $ touch pr1.sh
rigazizova@dk6n65 ~ $ emacs &
[1] 7815
```

Рис. 3.1: Создание файла

Используя команды `getopts` `grep`, запишем в файл программу (рис.2), которая анализирует командную строку с ключами: – `-iinputfile` — прочитать данные из указанного файла; – `-ooutputfile` — вывести данные в указанный файл; – `-r` 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`.

```
#!/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
```

U:*** pr1.sh All L34 (Shell-script[sh]) Чт мая 12 13:59 0.62

Рис. 3.2: Командный файл

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). Запускаем файл (рис.4).

```
rigazizova@dk6n65 ~ $ touch pr1.sh
rigazizova@dk6n65 ~ $ emacs &
[1] 7815
rigazizova@dk6n65 ~ $ chmod +x *.sh
[1]+  Завершён      emacs
rigazizova@dk6n65 ~ $ chmod +x *.sh
rigazizova@dk6n65 ~ $ touch one.txt
rigazizova@dk6n65 ~ $ touch two.txt
rigazizova@dk6n65 ~ $ mcedit one.txt
rigazizova@dk6n65 ~ $ mcedit two.txt
```

Рис. 3.3: Права доступа


```

rigazizova@dk6n65 ~ $ ./pr1.sh -i one.txt -o two.txt -p кот -C -n
rigazizova@dk6n65 ~ $ cat one.txt
Я видел чудное мгновение
Передо мной явилась ты
Как мимолетное видение
Как гений чистой красоты
rigazizova@dk6n65 ~ $ cat two.txt

```

Рис. 3.4: Выполнение

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 (рис.5)

```

emacs@dk4n58
File Edit Options Buffers Tools Sh-Script Help
^
#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;
}

U:--- pr2.c All L10 (C/*l Abbrev) Чт мая 12 14:20 0.2
^
#!/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

```

Рис. 3.5: Программы

Проверим работу командного файла, передав ему права на выполнения и

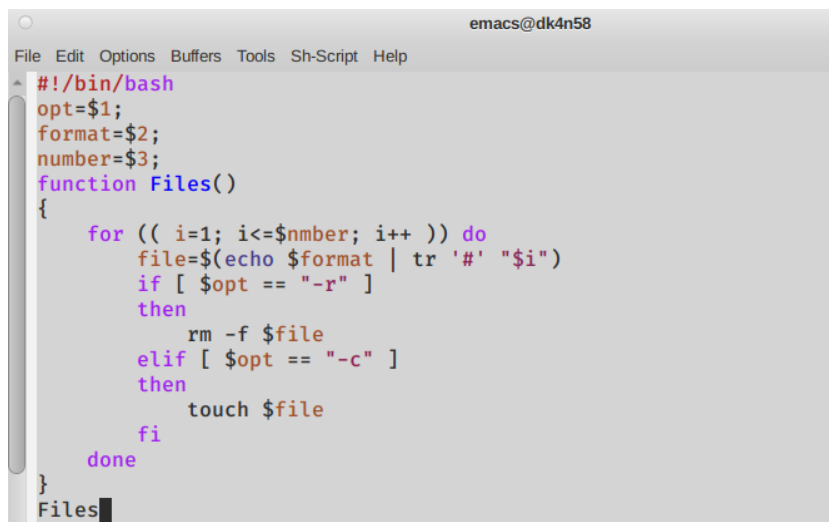
запустив его (рис.6).

```
rigazizova@dk6n65 ~ $ chmod +x pr2.sh
rigazizova@dk6n65 ~ $ ./pr2.sh
Введите число
-2
Число меньше 0
```

Рис. 3.6: Выполнение

3. Создадим командный файл files.sh и откроем его в emacs.

Напишем командный файл (рис.7), создающий указанное число файлов, пронумерованных последовательно от 1 до ∞ (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).



```
#!/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
```

Рис. 3.7: Командный файл

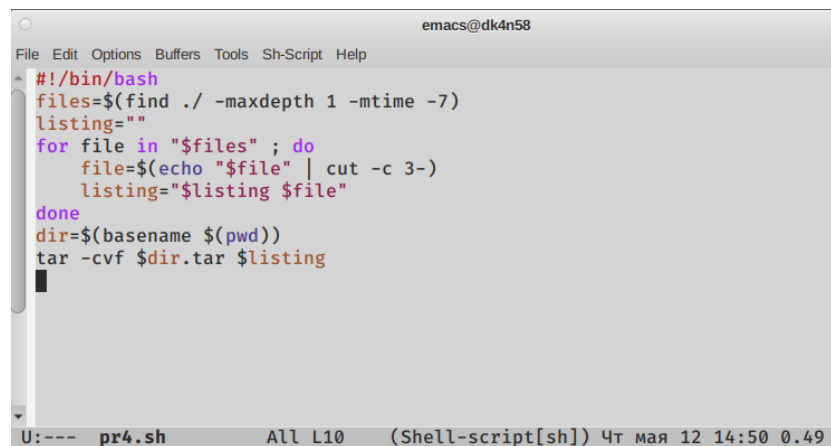
Проверим его работу, передав ему права на выполнения и запустив его (команда ./files.sh) (рис.8)

```
rigazizova@dk6n65 ~ $ ./files.sh -c et#.txt 4
rigazizova@dk6n65 ~ $ ls
abcl          backup.sh~  et3.txt    format.sh  lab09.sh   one.txt     pr1.sh~    proglis.sh~  R          tmp          Загрузки
Architecture_PC bin          et4.txt    format.sh~ lab2.1.pub  OS_2022    pr2        prog.sh      reports    two.txt     Изображен
australia     conf.txt    feathers   GNUstep    may         play        pr2.c      prog.sh~    rer.txt    work         Музыка
backup        et1.txt    files.sh   katalog     monthly    playy       pr2.sh     public      sk1.places Видео
backup.sh     et2.txt    file.txt  'lab09.sh' my_os       pr1.sh     proglis.sh public.html test.txt   Документы  Рабочий с
```

Рис. 3.8: Выполнение

4. Создадим командный файл pr4.sh и откроем его в emacs.

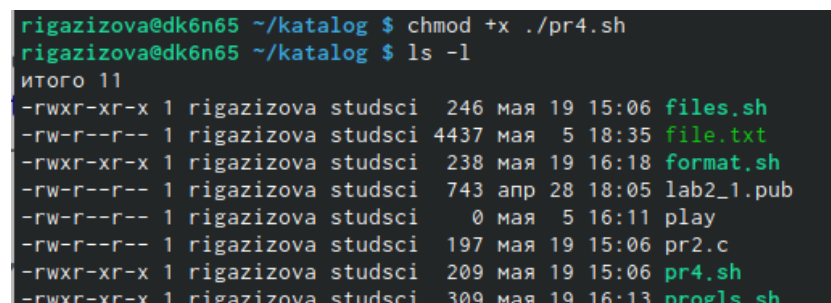
Напишем командный файл (рис.9), который с помощью команды tar запаковывает в архив все файлы в указанной директории. Модифицируем его так, чтобы запаковывались только те файлы, которые были изменены менее недели тому назад (использовать команду find).



```
emacs@dk4n58
File Edit Options Buffers Tools Sh-Script Help
#!/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
U:--- pr4.sh All L10 (Shell-script[sh]) Чт мая 12 14:50 0.49
```

Рис. 3.9: Командный файл

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 (рис.10). Файл работает исправно.



```
rigazizova@dk6n65 ~/katalog $ chmod +x ./pr4.sh
rigazizova@dk6n65 ~/katalog $ ls -l
итого 11
-rwxr-xr-x 1 rigazizova studsci 246 мая 19 15:06 files.sh
-rw-r--r-- 1 rigazizova studsci 4437 мая 5 18:35 file.txt
-rwxr-xr-x 1 rigazizova studsci 238 мая 19 16:18 format.sh
-rw-r--r-- 1 rigazizova studsci 743 апр 28 18:05 lab2_1.pub
-rw-r--r-- 1 rigazizova studsci 0 мая 5 16:11 play
-rw-r--r-- 1 rigazizova studsci 197 мая 19 15:06 pr2.c
-rwxr-xr-x 1 rigazizova studsci 209 мая 19 15:06 pr4.sh
-rwxr-xr-x 1 rigazizova studsci 309 мая 19 16:13 progl.s.sh
```

Рис. 3.10: Выполнение

4 Выводы

Я научилась писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

5 Контрольные вопросы

1). Команда `getopts` осуществляет синтаксический анализ командной строки, выделяя флаги, 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, `1.1 echo` – выведет имена всех файлов текущего каталога, что представляет собой простейший аналог ко-

манды `ls`; 1.2. `ls.c` – выведет все файлы с последними двумя символами, совпадающими с.с. 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`, ко-

торая 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 mans/i.s, mans/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.